DATA SCIENCE IN PYTHON

Natural Language Processing

★★★★★ *With Expert Data Science Instructor Alice Zhao*

MAVEN ANALYTICS

ABOUT THIS SERIES

This is **Part 5** of a **5-Part series** designed to take you through several applications of data science using Python, including **data prep & EDA**, **regression**, **classification**, **unsupervised learning** & **NLP**

PART 1
Data Prep & EDA

PART 2
Regression

PART 3
Classification

PART 4
Unsupervised Learning

PART 5
Natural Language Processing

COURSE STRUCTURE

This course is for students looking for a **practical, hands-on** approach to learning and applying natural language processing (NLP) concepts and techniques with Python

Additional resources include:

- **Downloadable PDF** to serve as a helpful reference when you're offline or on the go
- **Quizzes** & **Assignments** to test and reinforce key concepts, with step-by-step solutions
- **Interactive demos** to keep you engaged and apply your skills throughout the course

COURSE OUTLINE

1 Installation & Setup	*Install Anaconda, launch Jupyter Notebook to write Python code, and practice creating and activating conda environments*
2 Natural Language Processing 101	*Review the history of NLP, traditional ML vs modern LLM approaches, common NLP applications and Python libraries*
3 Text Preprocessing	*Walk through the NLP text preprocessing pipeline, including cleaning, normalization, linguistic analysis and vectorization*
4 NLP with Machine Learning	*Use traditional ML techniques to apply rule-based, supervised learning and unsupervised learning techniques on text data*

COURSE OUTLINE

5	**Neural Networks & Deep Learning**	*Understand the theory behind how neural networks and deep learning work before moving on to modern DL architectures*
6	**Transformers & LLMs**	*Dive into the main parts of a transformer, including embeddings, attention and FFNs, as well as popular LLMs (BERT, GPT, etc.)*
7	**Hugging Face Transformers**	*Use pretrained LLMs for sentiment analysis, NER, zero-shot classification, summarization, feature extraction and generation*
8	**NLP Review & Next Steps**	*Review the NLP techniques covered in this course, when to use them, and next steps to dive deeper and stay up-to-date*

THE COURSE ASSIGNMENTS

You'll be using **text cleaning, normalization and vectorization** techniques on book descriptions to extract common words and insights about the books

You'll be using **machine learning techniques** on movie summaries to rank movies by sentiment, predict the gender of a director, and identify movie themes

You'll be using **pretrained LLMs** on book descriptions to extract character names, classify books into categories, create summaries, and recommend similar books

SETTING EXPECTATIONS

This course covers **traditional & modern** natural language processing (NLP)

- *Traditional NLP includes text preprocessing techniques & machine learning algorithms for text data*
- *Modern NLP includes concepts like neural networks, deep learning, transformers, and large language models (LLMs)*

We will use **Anaconda** as our package and environment manager

- *Anaconda is free to download, and the industry standard for conducting data science tasks with Python*

We will use **Hugging Face** to work with Large Language Models (LLMs)

- *We'll use the Model Hub to access pretrained models and the Transformers library in Python to apply them*
- *We will NOT be doing a deep dive into more advanced transformer topics like fine-tuning, RAGs, etc.*

You do **NOT** need to be a Python expert to take this course

- *It is strongly recommended that you complete the first course in this series, Data Prep & EDA, but we will teach the relevant math and Python code for applying NLP techniques throughout this course*

INSTALLATION & SETUP

INSTALLATION & SETUP

In this section we'll install **Anaconda**, start writing Python code in a **Jupyter Notebook**, and learn how to create a new **conda environment** to get set up for this course

TOPICS WE'LL COVER:

- Anaconda Overview
- Installing Anaconda
- Launching Jupyter
- Conda Environments

GOALS FOR THIS SECTION:

- Learn about Anaconda's various features
- Install Anaconda and launch Jupyter Notebook
- Get comfortable writing Python code within the Jupyter Notebook interface
- Understand how to create and use conda environments for project organization

ANACONDA

- Anaconda Overview
- Installing Anaconda
- Launching Jupyter
- Conda Environments

Anaconda is the most popular package and environment manager for data science and machine learning tasks

When you install Anaconda, it comes with the following:

Coding languages & tools

*We'll be using **Jupyter Notebook** to write **Python** code*

Popular packages

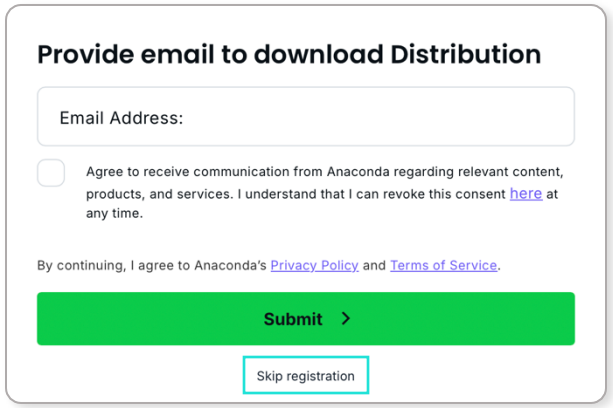
Package & environment manager

We highly recommend using **Anaconda** as your package and environment manager to follow along with the course demos

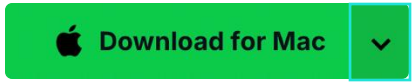
You can use pip installs for packages and the venv module for environments as an alternative if you're already familiar with them

INSTALL ANACONDA (MAC)

1) Go to **anaconda.com/download** and click "Skip registration"

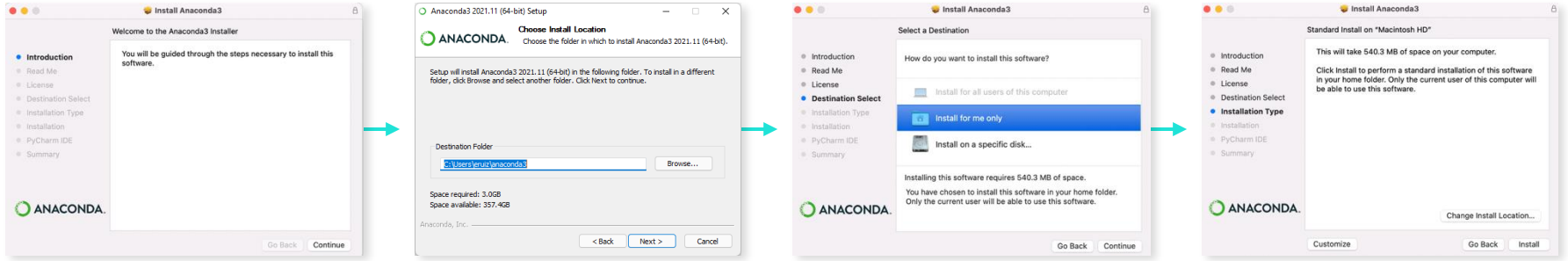


2) Click on the arrow in the "Download for Mac" button and select the type of computer you have *(Apple Silicon for newer computers, Intel for older ones)*



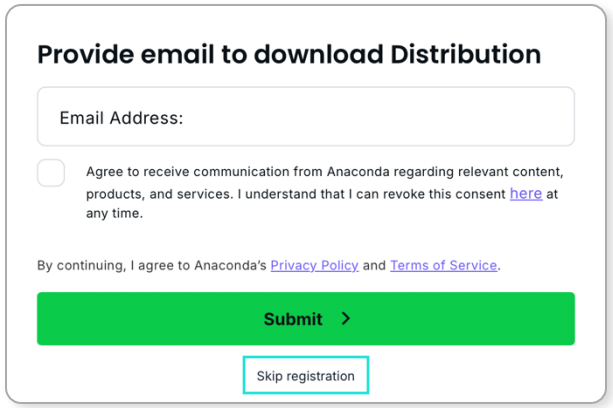
3) Launch the downloaded Anaconda **pkg** file

4) Follow the **installation steps** *(default settings are OK)* and click "Continue", "Agree" and "Install" at the end



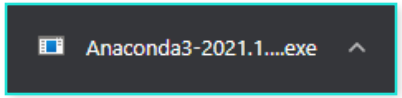
INSTALL ANACONDA (PC)

1) Go to **anaconda.com/download** and click "Skip registration"

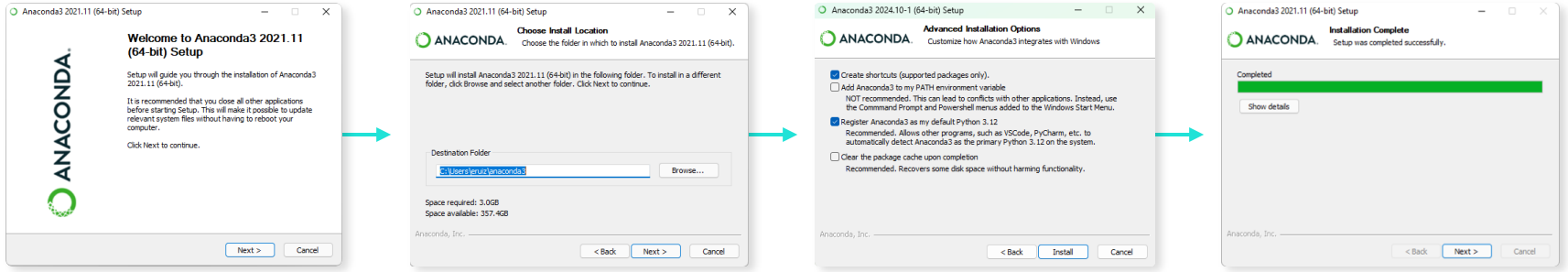


2) Click on "Download" button

3) Launch the downloaded Anaconda **exe** file



4) Follow the **installation steps** *(default settings are OK)* and click "Continue", "Agree" and "Install" at the end

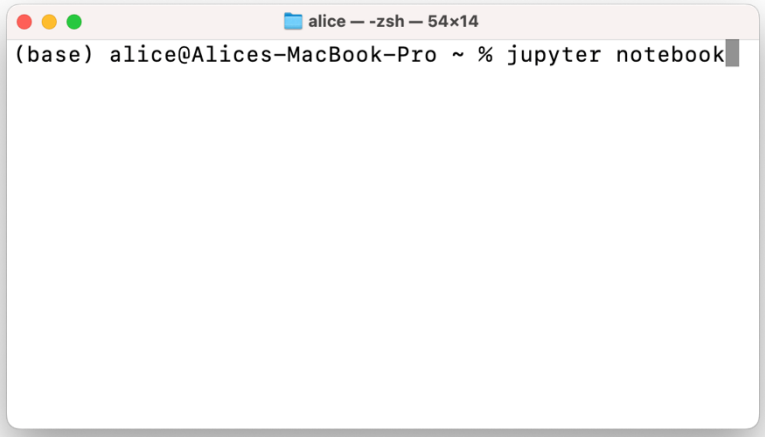


LAUNCH JUPYTER

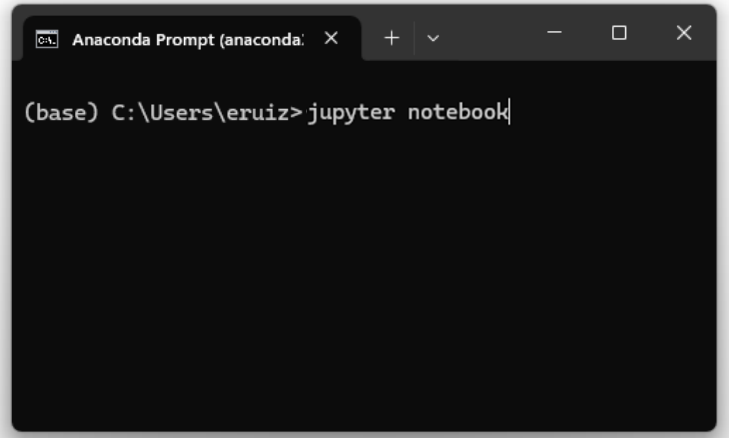
- Anaconda Overview
- Installing Anaconda
- Launching Jupyter
- Conda Environments

1) Open the **Terminal** (Mac) or **Anaconda Prompt** (PC) application
2) Type **jupyter notebook** and hit return

MAC

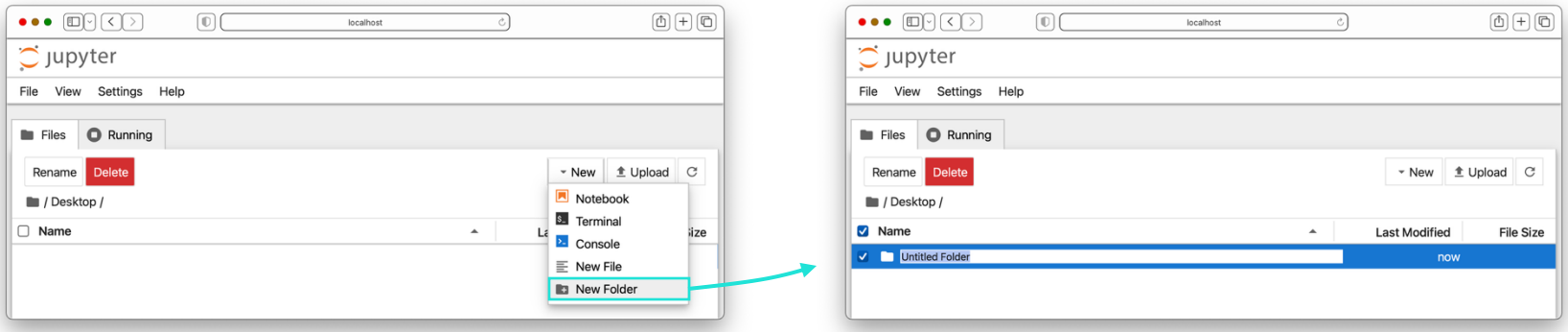


PC



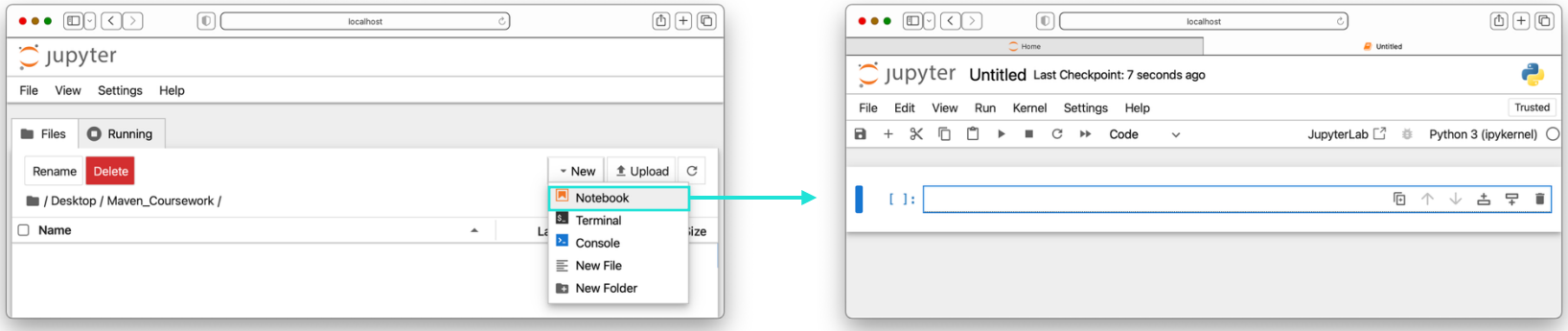
YOUR FIRST JUPYTER NOTEBOOK

1) Once inside the Jupyter interface, **create a folder** to store your notebooks for the course



NOTE: *You can rename the folder by clicking "Rename" in the top left corner*

2) Open your new coursework folder and **launch your first Jupyter Notebook**!

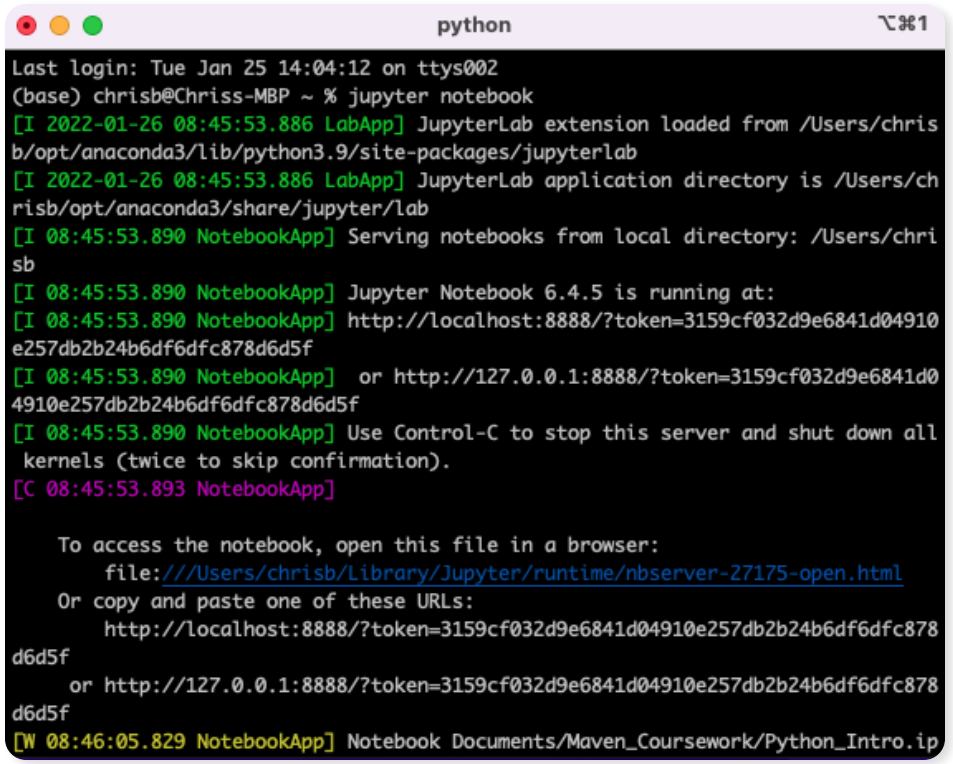


NOTE: *You can rename the notebook by clicking on the title at the top of the screen*

THE NOTEBOOK SERVER

- Anaconda Overview
- Installing Anaconda
- Launching Jupyter
- Conda Environments

NOTE: When you launch a Jupyter Notebook, you'll see a bunch of log data; this is called a **notebook server**, and it powers the notebook interface



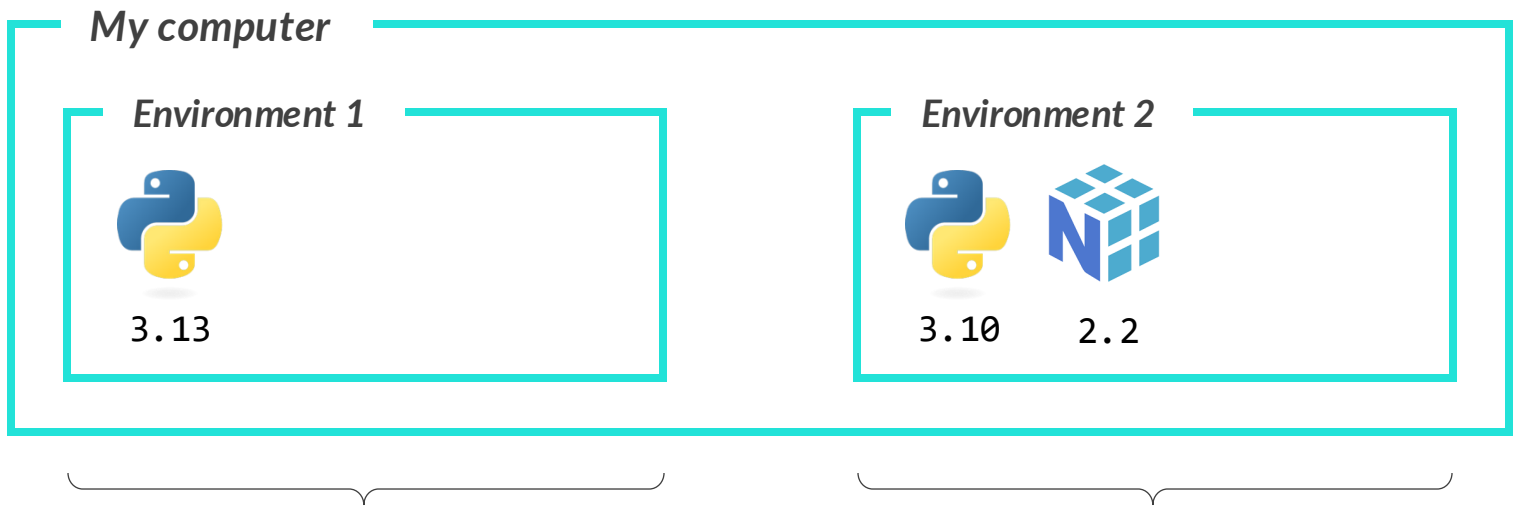
If you close the server window, **your notebooks will not run!**

Depending on your OS, and method of launching Jupyter, you may not see this – as long as you can run your notebooks, don't worry!

CONDA ENVIRONMENTS

- Anaconda Overview
- Installing Anaconda
- Launching Jupyter
- Conda Environments

A **conda environment** is a place on your computer where you can install specific versions of Python and Python packages without affecting other projects



I'm working on a beginner Python 101 project and am learning about built-in Python functions

*I'm going to **activate Environment 1** and do my Python coding here*

I'm taking a NumPy course where the instructor is using Python 3.10 with NumPy 2.2, and I want my code to match his

*I'm going to **activate Environment 2** and do my Python coding here*

CONDA ENVIRONMENTS

- Anaconda Overview
- Installing Anaconda
- Launching Jupyter
- Conda Environments

A **conda environment** is a place on your computer where you can install specific versions of Python and Python packages without affecting other projects

Environment 1

3.13

```
2 + 2
```

```
4
```

```
max([1, 2, 3, 4, 5])
```

```
5
```

```
import numpy as np
```

```
---------------------------
ModuleNotFoundError
```

We can use all built-in Python 3.13 functions, but get an error here because the NumPy library isn't available in this environment

Environment 2

3.10

2.2

```
2 + 2
```

```
4
```

```
max([1, 2, 3, 4, 5])
```

```
5
```

```
import numpy as np
```

```
np.sqrt(25)
```

```
5.0
```

We can use all built-in Python 3.10 functions, and we're able to import the NumPy library because it's installed in this environment

DEFAULT VS. NEW ENVIRONMENTS

As a Python beginner, you've likely been using the **default environment**, but advanced users create **new environments** for each new, complex project

- Creating a new environment gives us a blank slate to freshly install Python packages and make sure the versions and dependencies are correct for each project

Default environment

If you're just using a few basic libraries, using the default environment for all your projects is fine

New environment for sentiment analysis project

New environment for LLM project

CONDA WORKFLOW

Anaconda Overview

Installing Anaconda

Launching Jupyter

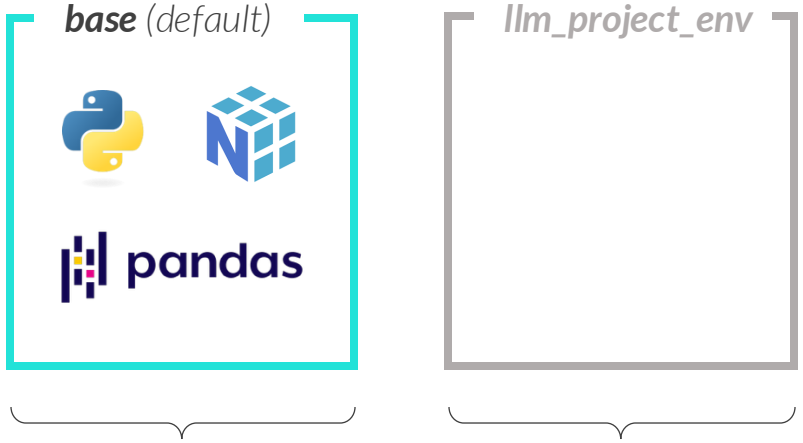
Conda Environments

This is the **workflow** for working with conda environments and packages:

EXAMPLE | *Creating a new environment to use Hugging Face's Transformers library*

1. **Create a new environment**
2. Activate the new environment
3. Install the packages you need
4. Launch Jupyter within this environment
5. Start writing Python code as usual
6. Deactivate the environment

```
> conda create --name llm_project_env
```



The base environment is active by default

A new, empty environment has been created called ***llm_project_env***

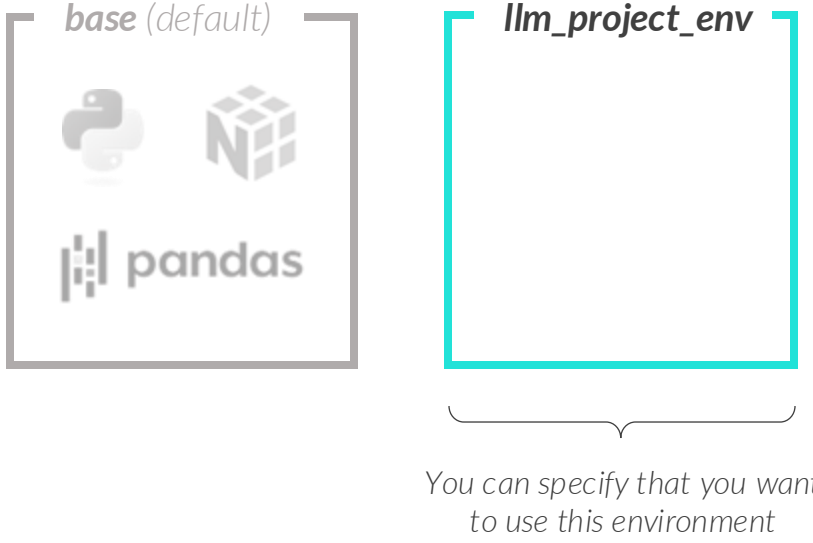
CONDA WORKFLOW

This is the **workflow** for working with conda environments and packages:

EXAMPLE | *Creating a new environment to use Hugging Face's Transformers library*

1. Create a new environment
2. **Activate the new environment**
3. Install the packages you need
4. Launch Jupyter within this environment
5. Start writing Python code as usual
6. Deactivate the environment

```
> conda activate llm_project_env
```



You can specify that you want to use this environment

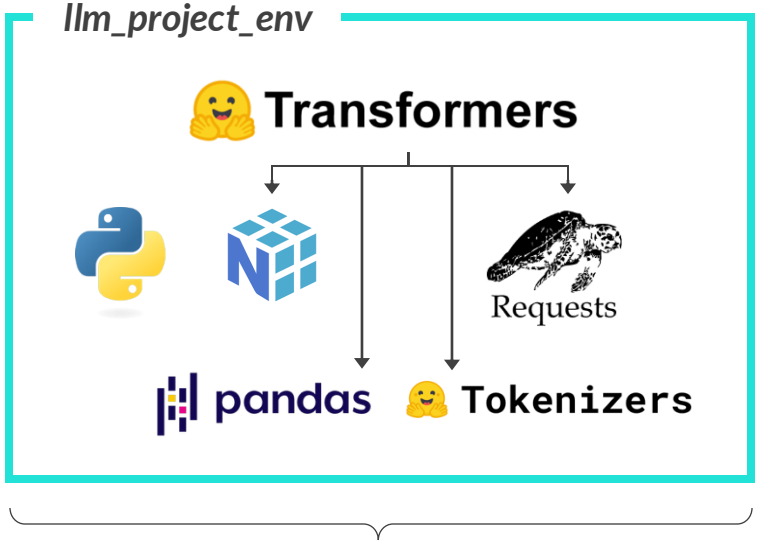
CONDA WORKFLOW

This is the **workflow** for working with conda environments and packages:

EXAMPLE | *Creating a new environment to use Hugging Face's Transformers library*

1. Create a new environment
2. Activate the new environment
3. **Install the packages you need**
4. Launch Jupyter within this environment
5. Start writing Python code as usual
6. Deactivate the environment

```
> conda install transformers
```

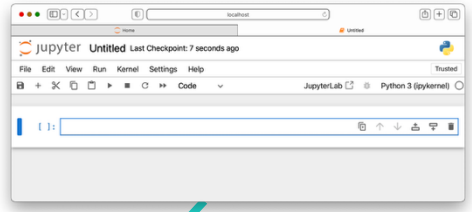


With just one line of code, the Transformers package is installed along with its dependencies, which are other packages that it uses code from

CONDA WORKFLOW

This is the **workflow** for working with conda environments and packages:

EXAMPLE | *Creating a new environment to use Hugging Face's Transformers library*



1. Create a new environment
2. Activate the new environment
3. Install the packages you need
4. **Launch Jupyter within this environment**
5. Start writing Python code as usual
6. Deactivate the environment

```
> jupyter notebook
```

llm_project_env

Transformers

Requests

pandas

Tokenizers

The Jupyter Notebook you open will have access to the packages available in the active environment

CONDA WORKFLOW

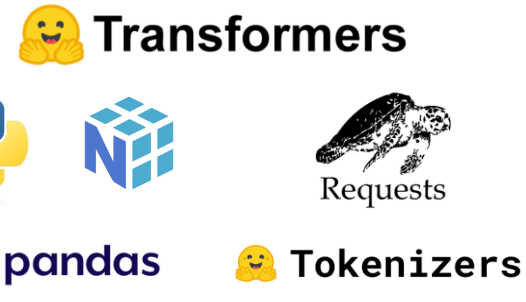
This is the **workflow** for working with conda environments and packages:

EXAMPLE | *Creating a new environment to use Hugging Face's Transformers library*

1. Create a new environment
2. Activate the new environment
3. Install the packages you need
4. Launch Jupyter within this environment
5. **Start writing Python code as usual**
6. Deactivate the environment

```python
from transformers import pipeline

sentiment = pipeline("sentiment-analysis")
sentiment("I love NLP!")
```

llm_project_env



The code uses the packages in the active environment

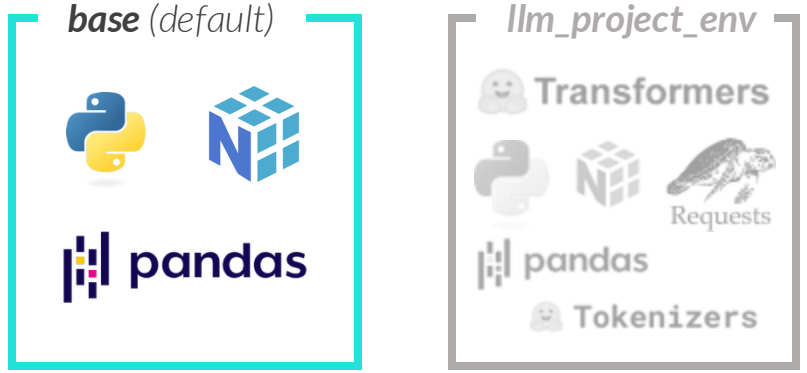
CONDA WORKFLOW

This is the **workflow** for working with conda environments and packages:

EXAMPLE | *Creating a new environment to use Hugging Face's Transformers library*

1. Create a new environment
2. Activate the new environment
3. Install the packages you need
4. Launch Jupyter within this environment
5. Start writing Python code as usual
6. **Deactivate the environment**

```
> conda deactivate
```



Deactivating takes you back to the base environment

CONDA COMMANDS

These are some helpful **commands** when working with conda environments:

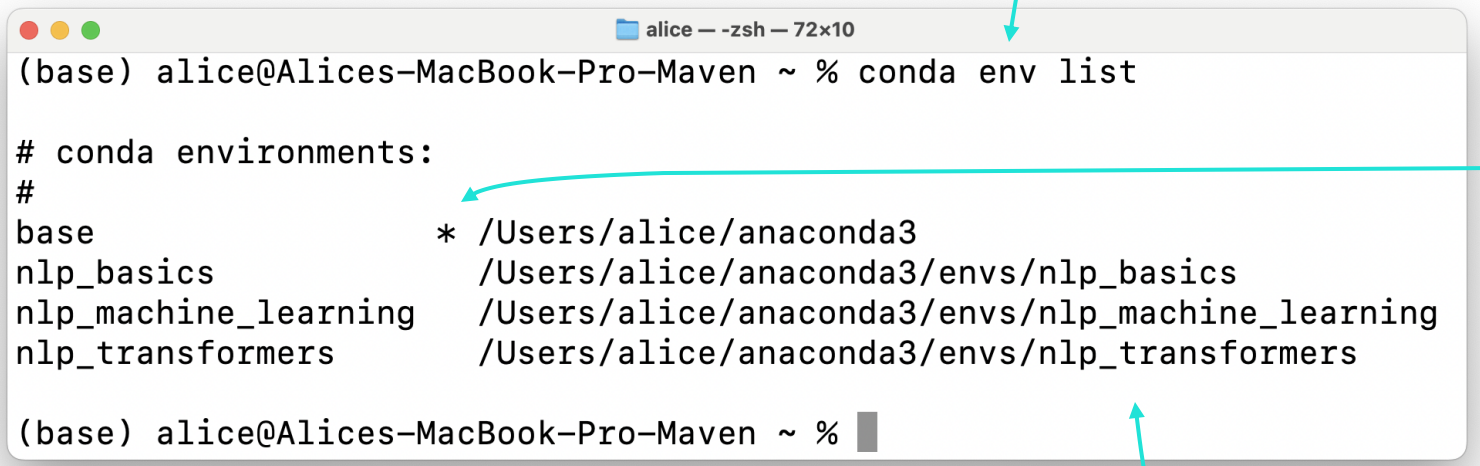
Category	Command	Description
Environment	conda env list	Lists all conda environments in your system
	conda create --name *test_env*	Creates a new environment called *text_env*
	conda activate *test_env*	Activates the *test_env* environment
	conda deactivate	Deactivates the current environment and returns to the base environment
Package	conda list	Lists installed packages in the active environment
	conda install	Installs specified packages into the active environment
YML	conda env export > *text_env.yml*	Exports package names and versions in the active environment into a .yml file
	conda env create -f *test_env.yml*	Create a new environment from a .yml file

CONDA COMMANDS

All **conda commands** should be written and executed within the Terminal (Mac) or Anaconda Prompt (PC) application

The (base) prefix tells us we're in the default environment

"conda env list" is the conda command to display all the available environments



*The * signals the active environment*

These are the three NLP environments we'll be creating throughout this course

You will only see the (base) prefix if you have Anaconda installed

ENVIRONMENTS IN THIS COURSE

We will be creating and using **four conda environments** throughout this course

- While you can still complete the course without utilizing environments, they will help keep you organized and avoid potential version conflicts

Section	Environment
1) Installation & Setup	test_env
2) Natural Language Processing 101	
3) Text Preprocessing	nlp_basics
4) NLP with Machine Learning	nlp_machine_learning
5) Neural Networks & Deep Learning	
6) Transformers & LLMs	
7) Hugging Face Transformers	nlp_transformers
8) NLP Review & Next Steps	

If you have experience working with .yml files, you can find the NLP environment .yml files in the "Environments" folder within the course resources

You can use them as reference or to quickly create new conda environments

NATURAL LANGUAGE PROCESSING 101

NATURAL LANGUAGE PROCESSING 101

In this section we'll cover the basics of **natural language processing (NLP)**, including key concepts, the evolution of NLP over the years, and its applications & Python libraries

TOPICS WE'LL COVER:

- NLP Basics
- History of NLP
- Techniques & Applications

GOALS FOR THIS SECTION:

- Understand the basics of NLP
- Learn how NLP has evolved over the years and the techniques that are commonly used today
- Become familiar with the variety of applications, techniques, and Python libraries available for NLP

NLP BASICS

- NLP Basics
- History of NLP
- Techniques & Applications

NATURAL LANGUAGE PROCESSING

noun

The ~~application of machine learning algorithms to the analysis, understanding, and manipulation of written or spoken examples of human language*~~

Using computers to work with text data

Customer	Rating	Review
Remy	5	The food at this restaurant was amazing!!
Anton	3.5	Long wait. The customer service was meh.
Colette	4.5	Great ambiance and drinks. Would come back!

It's easy for humans to read & interpret these reviews, but computers struggle; ***that's where NLP comes in!***

*Dictionary.com

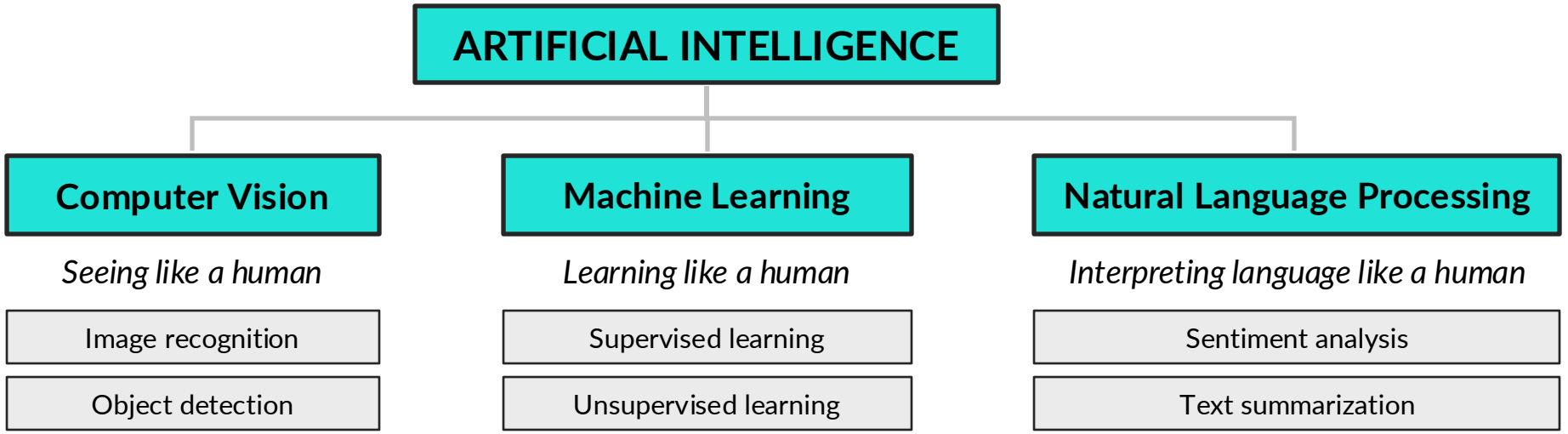
NLP & AI

NLP Basics

History of NLP

Techniques & Applications

Natural Language Processing falls under **Artificial Intelligence (AI)**, which is a field that tries to replicate what humans can do using computers



Where does data science fit in here?

- Data scientists use data to extract insights, and they can do so by applying and interpreting various computer vision, machine learning, and natural language processing models

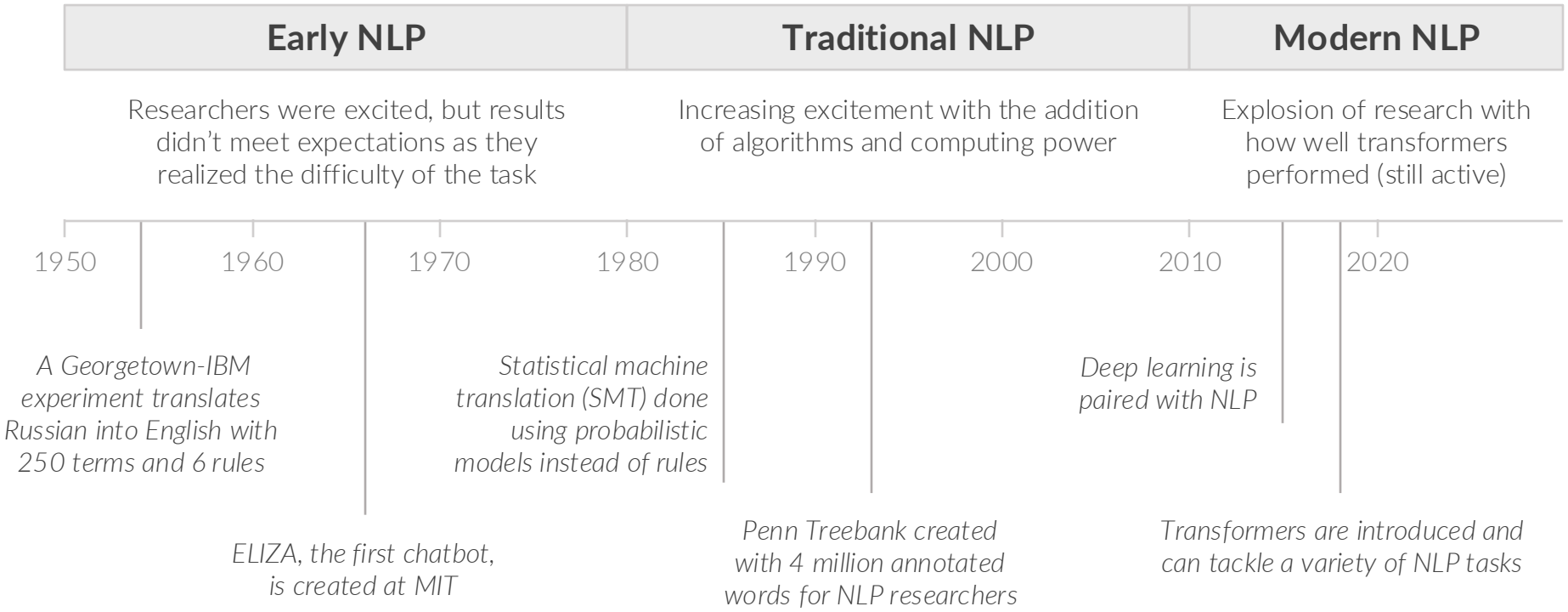
HISTORY OF NLP

The field of NLP has evolved significantly over the past 70+ years:

NLP Basics

History of NLP

Techniques & Applications



HISTORY OF NLP

The field of NLP has evolved significantly over the past 70+ years:

Early NLP	Traditional NLP	Modern NLP
Rules-based techniques • Grammar rules • Pattern matching	Statistical techniques • Statistics • Probability Machine learning techniques • Supervised learning • Unsupervised learning	Recurrent-based techniques • RNNs, LSTMs, etc. Transformer-based techniques • LLMs: GPT, BERT, etc.

HISTORY OF NLP

The field of NLP has evolved significantly over the past 70+ years:

NLP Basics

History of NLP

Techniques & Applications

Early NLP	Traditional NLP	Modern NLP
Rules-based techniques • Grammar rules • Pattern matching	Statistical techniques • Statistics • Probability	Recurrent-based techniques • RNNs, LSTMs, etc.
Traditional NLP has largely replaced Early NLP	Machine learning techniques • Supervised learning • Unsupervised learning	Transformer-based techniques • LLMs: GPT, BERT, etc.

HISTORY OF NLP

The field of NLP has evolved significantly over the past 70+ years:

NLP Basics

History of NLP

Techniques & Applications

Early NLP	Traditional NLP	Modern NLP
Rules-based techniques • Grammar rules • Pattern matching	Statistical techniques • Statistics • Probability Machine learning techniques • Supervised learning • Unsupervised learning	Recurrent-based techniques • RNNs, LSTMs, etc. Transformer-based techniques • LLMs: GPT, BERT, etc.

Transformer-based NLP has largely replaced Recurrent-based NLP

HISTORY OF NLP

The field of NLP has evolved significantly over the past 70+ years:

NLP Basics

History of NLP

Techniques & Applications

Early NLP	Traditional NLP	Modern NLP
Rules-based techniques • Grammar rules • Pattern matching	Statistical techniques • Statistics • Probability Machine learning techniques • Supervised learning • Unsupervised learning	Recurrent-based techniques • RNNs, LSTMs, etc. Transformer-based techniques • LLMs: GPT, BERT, etc.

There's a big mindset shift from traditional to modern NLP:

- With traditional NLP, there's a focus on **understanding** everything that's happening
- With modern NLP, there's no way to understand everything that's happening, and what matters most is **performance**, even if it's all a black box

NLP APPLICATIONS & TECHNIQUES

There are numerous **NLP applications & techniques** that we'll cover:

- NLP Basics
- History of NLP
- Techniques & Applications

NLP Category	Technique	Application
Traditional	Rules-Based	→ *Sentiment Analysis*
	Supervised Learning (Naïve Bayes)	→ *Text Classification*
	Unsupervised Learning (NMF)	→ *Topic Modeling*
Modern	Encoder-Only LLM (BERT)	→ *Sentiment Analysis* *Named Entity Recognition (NER)*
	Encoder-Decoder LLM (BART)	→ *Zero Shot Classification* *Text Summarization*
	Decoder-Only LLM (GPT)	→ *Text Generation*
	Embeddings (MiniLM)	→ *Document Similarity*

NLP LIBRARIES IN PYTHON

NLP Basics

History of NLP

Techniques & Applications

Most general data science tasks can be done using Pandas and Scikit-learn, but there are many available **Python libraries** for NLP tasks:

Course Section	Library	Applications
Text Preprocessing	Pandas	Cleaning & Normalization
	SpaCy	Cleaning & Normalization, Linguistic Analysis
	Scikit-learn	Vectorization
NLP with Machine Learning	VADER	Sentiment Analysis
	Scikit-learn	Text Classification, Topic Modeling
Neural Networks & Deep Learning	Scikit-learn	Classification & Regression
Hugging Face Transformers	Transformers	Text Summarization, Text Generation, etc.

These are other popular NLP libraries that we will NOT be covering as a part of this course (nltk, genism, TensorFlow, and PyTorch) as we will focus on the simplicity and ease of use

KEY TAKEAWAYS

Natural language processing allows computers to work with text data

- *NLP falls under the umbrella of Artificial Intelligence (AI), which is about making computers imitate human behaviors (like interpreting their natural language)*

NLP techniques have **greatly evolved** over the past 70+ years

- *Starting with rules-based techniques in the 1950s-70s, then moving onto traditional ML techniques in the 1980s-2000s, and currently modern NLP with deep learning and transformers-based techniques*

There can be **multiple approaches** to tackle various NLP problems

- *While transformers have been popularized for providing amazing results, simple rules-based or machine-learning based techniques are still important to understand for smaller to medium data sets*

Python is one of the best coding languages for applying NLP techniques

- *There are many NLP libraries, such as scikit-learn and transformers, which integrate well into other frameworks*

TEXT PREPROCESSING

TEXT PREPROCESSING

In this section we'll review the **text preprocessing** steps required before applying machine learning algorithms, including cleaning, normalization, vectorization, and more

TOPICS WE'LL COVER:

- NLP Pipeline
- Text Preprocessing with Pandas
- Text Preprocessing with spaCy
- Vectorization

GOALS FOR THIS SECTION:

- Learn the standard natural language processing workflow, also called the NLP pipeline
- Apply text cleaning and normalization techniques using Python's Pandas and spaCy libraries
- Understand how to format text data in a way that a computer can process using vectorization with word counts and TF-IDF scores

DATA SCIENCE WORKFLOW

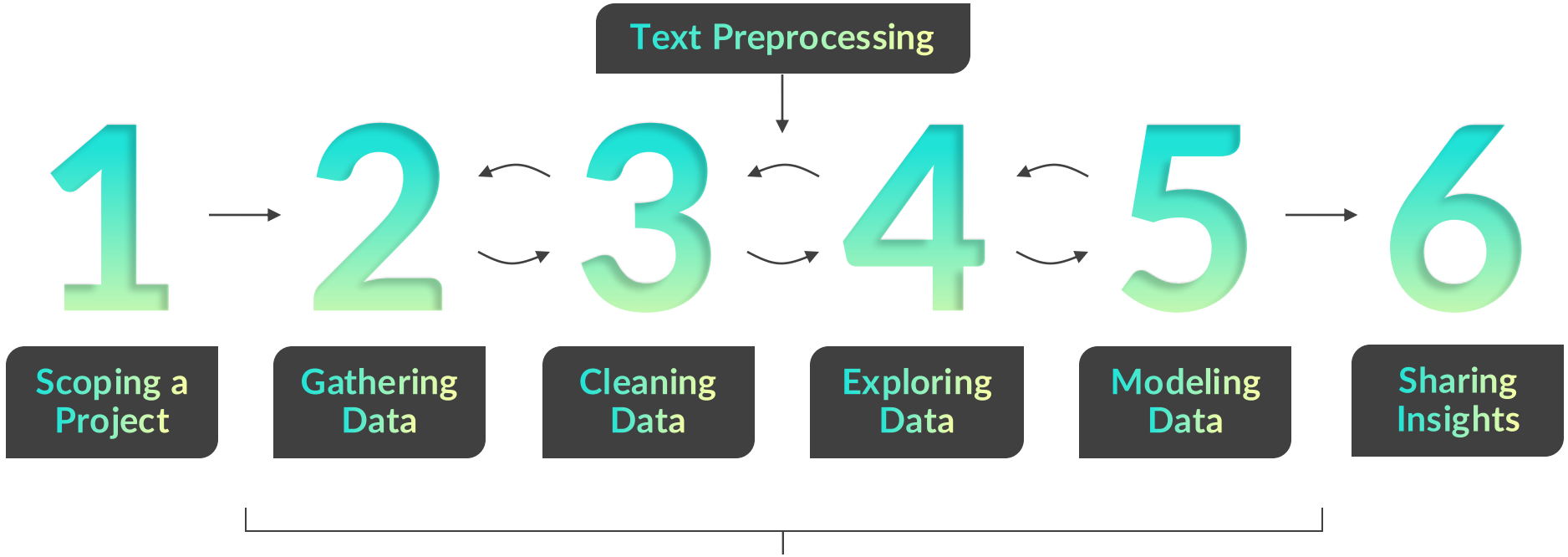
NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

NLP projects follow the same **data science workflow**, except there's an extra text preprocessing step between cleaning and exploring data:



*For NLP projects, this portion is called the **NLP pipeline**, which is the series of steps your text data goes through for processing and analysis*

*This workflow is discussed in more detail in the Data Prep & EDA course

TEXT PREPROCESSING

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Text preprocessing is about preparing raw text data for analysis and modeling:

Generally clean text data

Cleaning & Normalization

- **Cleaning:** remove unnecessary text
- **Normalization:** make text consistent
- These steps can be done using a combination of Pandas and spaCy

Vectorization

- Turn text into a matrix of numbers
- Each document is represented by a vector of counts or TF-IDF values
- This can be done with scikit-learn

Text data ready for EDA and modeling

TEXT PREPROCESSING TECHNIQUES

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

We'll be covering these **text preprocessing** techniques:

Category	Concepts	Description
Cleaning & Normalization*	Lowercasing	Convert all text to lowercase
	Special Characters	Remove punctuation & special characters using regular expressions
	Tokenization	Split text into smaller units, like words or sentences
	Stemming / Lemmatization	Reduce words to their root or base form
	Stop Words	Remove common, non-essential words
	Parts of Speech (POS) Tagging	Identify grammatical roles of words (nouns, verbs, etc.)
Vectorization	Document-Term Matrix (DTM)	Represent text by word frequency, also known as Bag of Words
	TF-IDF	Extension of DTM that weights words based on their importance

**These text cleaning and normalization steps can be mixed and matched*

ASSIGNMENT: CREATE A NEW ENVIRONMENT

NEW MESSAGE

May 15, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **NLP Onboarding**

Hi,

I hear you're the new associate data scientist on the team – welcome!

We're currently kicking off several natural language processing projects with our client, Maven Books.

I'd like to get you involved ASAP. Can you create a new conda environment on your computer, and install the latest versions of Python and any other NLP libraries you might need?

Talk soon!
Lexi

Reply Forward

Key Objectives

1. Open the Terminal (Mac) or Anaconda Prompt (PC) application and create a new conda environment called "nlp_basics"
2. Activate the "nlp_basics" environment
3. Install Python, Jupyter Notebook, Pandas, spaCy, Scikit-learn, and Matplotlib in the environment
4. Launch Jupyter within the environment
5. Write and execute a line of Python code

TEXT PREPROCESSING WITH PANDAS

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

The Pandas library is used for simple text cleaning and normalization

- Use **str.lower()** to make all text lowercase
- Use **str.replace()** to replace special characters *(punctuation, numbers, etc.)*

```
# lowercase text
df['sentence_clean'] = df['sentence'].str.lower()
df
```

	sentence	sentence_clean
0	When life gives you lemons, make lemonade! 🙂	when life gives you lemons, make lemonade! 🙂
1	She bought 2 lemons for $1 at Maven Market.	she bought 2 lemons for $1 at maven market.
2	A dozen lemons will make a gallon of lemonade. [AllRecipes]	a dozen lemons will make a gallon of lemonade. [allrecipes]

Everything is lowercase now!

TEXT PREPROCESSING WITH PANDAS

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

The Pandas library is used for simple text cleaning and normalization

- Use **str.lower()** to make all text lowercase
- Use **str.replace()** to replace special characters *(punctuation, numbers, etc.)*

```
# remove text between brackets, including the brackets
df['sentence_clean'] = df['sentence_clean'].str.replace(r'\[.*?\]', '', regex=True)
df
```

	sentence	sentence_clean
0	When life gives you lemons, make lemonade! 🙂	when life gives you lemons, make lemonade! 🙂
1	She bought 2 lemons for $1 at Maven Market.	she bought 2 lemons for $1 at maven market.
2	A dozen lemons will make a gallon of lemonade. [AllRecipes]	a dozen lemons will make a gallon of lemonade.

No text between brackets!

PRO TIP: Regular expressions (regex) allow you to find patterns; once you understand the basic concept, you can use tools like ChatGPT to generate the syntax

TEXT PREPROCESSING WITH PANDAS

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

The Pandas library is used for simple text cleaning and normalization

- Use **str.lower()** to make all text lowercase
- Use **str.replace()** to replace special characters *(punctuation, numbers, etc.)*

```
# remove punctuation
df['sentence_clean'] = df['sentence_clean'].str.replace(r'[^\w\s]', '', regex=True)
df
```

	sentence	sentence_clean
0	When life gives you lemons, make lemonade! 🙂	when life gives you lemons make lemonade
1	She bought 2 lemons for $1 at Maven Market.	she bought 2 lemons for 1 at maven market
2	A dozen lemons will make a gallon of lemonade. [AllRecipes]	a dozen lemons will make a gallon of lemonade

No more special characters!

PRO TIP: Regular expressions (regex) allow you to find patterns; once you understand the basic concept, you can use tools like ChatGPT to generate the syntax

ASSIGNMENT: TEXT PREPROCESSING WITH PANDAS

NEW MESSAGE

May 16, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **Text preprocessing request**

Hello,

Now that you're all settled in, let me get you up to speed with our first task for the Maven Books project: text preprocessing

I hear you're already familiar with Pandas. We've been given a flat file of the top 100 children's books over the past century.

Can you use Pandas string functions to do some text normalization and cleaning?

Thank you!
Lexi

childrens_books.csv

Reply Forward

Key Objectives

1. Read the childrens_books.csv file into a Jupyter Notebook
2. Within the Description column:
    a) Make all the text lowercase
    b) Remove all \xa0 characters
    c) Remove all punctuation

TEXT PREPROCESSING WITH SPACY

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

The **spaCy** library can handle many NLP tasks, including tokenization, lemmatization, stop words, and more

- The first step is to turn a text string into a spaCy doc object

```
# load the spacy english model
import spacy
nlp = spacy.load('en_core_web_sm')
```

When you import spaCy, you need to specify which language to use – in this case, we're choosing English, which includes information from a large amount of annotated text

```
# view a single phrase
phrase = 'im selling lemons for $5 today'
phrase
```

```
'im selling lemons for $5 today'
```

```
# create a spacy doc object
doc = nlp(phrase)
doc
```

```
im selling lemons for $5 today
```

Here we're converting a single string into a spaCy object

Now that doc (document) is a spaCy object, we can use all the available spaCy NLP methods on the text

TOKENIZATION

Tokenization lets you break text up into smaller units, like words

- Text strings are often split by whitespace to make tokens

```
doc
```

```
im selling lemons for $5 today
```

```
# break up the text into tokens
[token.text for token in doc]
```

```
['i', 'm', 'selling', 'lemons', 'for', '$', '5', 'today']
```

This [] syntax is called a list comprehension

The way to read it is, for every token in the document, return the token text

spaCy mainly splits on whitespace, but there's some additional, smarter logic:

- Common contractions are separated (I'm)
- Punctuation is typically separated unless it's a URL, email address, etc.
- ...and much more!

LEMMATIZATION

Lemmatization reduces words to their base form

- spaCy uses a combination of linguistic rules and statistical models to lemmatize text

```
doc
```

```
im selling lemons for $5 today
```

```
# lemmatize the tokens to their root form
[token.lemma_ for token in doc]
```

```
['I', 'm', 'sell', 'lemon', 'for', '$', '5', 'today']
```

With lemmatization:

- *"i" has been updated to "I"*
- *"selling" has been updated to "sell"*
- *"lemons" has been updated to "lemon"*

What's the difference between lemmatization and stemming?

- They both reduce words to their base form, but lemmatization is the smarter approach and generally performs better – when choosing one, go with lemmatization
- Stemming: *am → am, is → is, are → ar* *happy → happi, happiness → happi*
- Lemmatization: *am → be, is → be, are → be* *happy → happy, happiness → happy*

STOP WORDS

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Stop words are words without any significant meaning

- You can view the full stop word list in spaCy with the code print(nlp.Defaults.stop_words)

```
doc
```

```
im selling lemons for $5 today
```

```
# remove the stop words
norm = [token.lemma_ for token in doc if not token.is_stop]
norm
```

The logic here is to only return tokens that are not stop words

```
['m', 'sell', 'lemon', '$', '5', 'today']
```

Note that "I" and "for" have been removed

PARTS OF SPEECH TAGGING

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Parts of speech (POS) tagging lets you label nouns, verbs, etc. within text data

- This is optional, but is sometimes used as a filtering technique to only look at nouns and pronouns for analysis, for example

```
doc
```

```
im selling lemons for $5 today
```

```
# view the parts of speech tags
pos = [(token.text, token.pos_) for token in doc]
pos
```

```
[('i', 'PRON'),
 ('m', 'AUX'),
 ('selling', 'VERB'),
 ('lemons', 'NOUN'),
 ('for', 'ADP'),
 ('$', 'SYM'),
 ('5', 'NUM'),
 ('today', 'NOUN')]
```

This is a lesser used technique compared to the others and one of many linguistic analysis capabilities available within spaCy:

- Other types of linguistic analysis include Named Entity Recognition (NER), Dependency Parsing and more
- Linguistic analysis techniques work better with raw text
- spaCy uses a combination of linguistic rules and statistical models for linguistic analysis

ASSIGNMENT: TEXT PREPROCESSING WITH SPACY

NEW MESSAGE

May 19, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **RE: Text preprocessing request**

Hi again,

Thanks for the first round of text preprocessing you did earlier with Pandas!

Could you do a second round of normalization and cleaning on the Description column with spaCy to tokenize, lemmatize and remove stop words from the text?

Thank you!
Lexi

Reply Forward

Key Objectives

1. In addition to the lowercasing and special character removal from the previous assignment, within the cleaned Description column:
    a) Tokenize the text
    b) Lemmatize the text
    c) Remove stop words

VECTORIZATION

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Vectorization is the process of converting text data into numeric data so that future data analysis and machine learning techniques can be applied

- Most ML techniques require text data to be cleaned, normalized and in a numeric format
- Some techniques, such as sentiment analysis, require text data to be in its raw text form

We will be covering these **vectorization techniques**:

Word Counts

lemon	lemonade	life	market	maven
1	1	1	0	0
1	0	0	1	1
1	1	0	0	0
6	0	0	0	0
1	0	0	1	0
2	0	0	1	1
0	1	0	0	0
0	0	0	0	0

TF-IDF

lemon	lemonade	life	market	maven
0.375318	0.543168	0.75107	0.000000	0.000000
0.411442	0.000000	0.00000	0.595449	0.690041
0.300100	0.434311	0.00000	0.000000	0.000000
1.000000	0.000000	0.00000	0.000000	0.000000
0.300100	0.000000	0.00000	0.434311	0.000000
0.556591	0.000000	0.00000	0.402755	0.466736
0.000000	0.364907	0.00000	0.000000	0.000000
0.000000	0.000000	0.00000	0.000000	0.000000

Embeddings

	dim1	dim2	dim3	dim768
0	-0.683818	0.707803	-0.252605	0.907771
1	0.858589	0.681373	-0.915621	-0.243296
2	0.977248	0.382527	0.165179	0.602443
3	-0.646722	-0.017643	0.811024	-0.352068
4	-0.599356	0.982837	0.753754	-0.195379
5	-0.944077	0.400329	0.837143	0.275571
6	-0.993053	0.069468	0.409714	0.428302
7	-0.187547	-0.746445	0.856585	-0.711975

(We'll cover this in modern NLP later!)

DOCUMENT-TERM MATRIX

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Clean, normalized text is turned into a **Document-Term Matrix** (DTM) for vectorization

- Each row represents a document, and each column represents a term
- The values within the DTM can be word counts, TF-IDF scores, or other calculated values

```
text_clean

0                      life lemon lemonade
1                      lemon maven market
2              dozen lemon gallon lemonade
3    lemon lemon lemon lemon lemon lemon
4                s market lemon sale today
5          maven market eureka lemon lemon
6             palmer lemonade half ice tea
7                          ice tea favorite
Name: sentence, dtype: object
```

	ice	lemon	lemonade	market	maven	tea
0	0	1	1	0	0	0
1	0	1	0	1	1	0
2	0	1	1	0	0	0
3	0	6	0	0	0	0
4	0	1	0	1	0	0
5	0	2	0	1	1	0
6	1	0	1	0	0	1
7	1	0	0	0	0	1

In this example, every value is the count of each term (columns) in each document (rows)

A DTM is a **bag of words** representation of text, where each document is represented by how often certain words appear, regardless of word order

COUNT VECTORIZER IN PYTHON

- NLP Pipeline
- Text Preprocessing with Pandas
- Text Preprocessing with spaCy
- Vectorization

Create a **Count Vectorizer** object to make a Document-Term Matrix in Python

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=2)
```

Language to remove stop words for (default is None)

Range for the sequence of "n" words to consider as a term in the DTM

Examples:

- *(1,1) – "data" (default)*
- *(1,2) – "data", "data science"*
- *(3,3) – "data science workflow"*

Number of OR percent of documents a term needs to appear in to be included in the DTM (default is 1)

You'll notice that we're able to **tokenize and remove stop words** using both **spaCy AND sklearn**, so it's your choice with which library you choose to do so

COUNT VECTORIZER IN PYTHON

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Create a **Count Vectorizer** object to make a Document-Term Matrix in Python

```
text_clean
```

```
0                    life lemon lemonade
1                     lemon maven market
2             dozen lemon gallon lemonade
3    lemon lemon lemon lemon lemon lemon
4               s market lemon sale today
5         maven market eureka lemon lemon
6            palmer lemonade half ice tea
7                       ice tea favorite
Name: sentence, dtype: object
```

```
# basic count vectorizer code
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
dtm = cv.fit_transform(text_clean)
dtm_df = pd.DataFrame(dtm.toarray(), columns=cv.get_feature_names_out())
dtm_df
```

	dozen	eureka	favorite	gallon	half	ice	lemon	lemonade	life	market	maven	palmer	sale	tea	today
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0
2	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1
5	0	1	0	0	0	0	2	0	0	1	1	0	0	0	0
6	0	0	0	0	1	1	0	1	0	0	0	1	0	1	0
7	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0

With the default parameters, these are the word counts for the 15 terms (columns) across the 8 documents (rows)

COUNT VECTORIZER IN PYTHON

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Create a **Count Vectorizer** object to make a Document-Term Matrix in Python

```
text_clean

0                    life lemon lemonade
1                     lemon maven market
2              dozen lemon gallon lemonade
3    lemon lemon lemon lemon lemon lemon
4                s market lemon sale today
5          maven market eureka lemon lemon
6             palmer lemonade half ice tea
7                        ice tea favorite
Name: sentence, dtype: object
```

```
# count vectorizer code with parameter tweaks
cv2 = CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=2)
dtm2 = cv2.fit_transform(text_clean)
dtm_df2 = pd.DataFrame(dtm2.toarray(), columns=cv2.get_feature_names_out())
dtm_df2
```

	ice	ice tea	lemon	lemon lemon	lemonade	market	maven	maven market	tea
0	0	0	1	0	1	0	0	0	0
1	0	0	1	0	0	1	1	1	0
2	0	0	1	0	1	0	0	0	0
3	0	0	6	5	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	0	0	2	1	0	1	1	1	0
6	1	1	0	0	1	0	0	0	1
7	1	1	0	0	0	0	0	0	1

With these parameters, we're removing all English stop words, returning all one and two-word terms, and keeping terms that appear in 2 or more documents

The columns have been reduced from the original 15 to 9!

ASSIGNMENT: COUNT VECTORIZER

NEW MESSAGE

May 20, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **Vectorization request**

Hello,

Now that you've cleaned and normalized the book descriptions using pandas and spaCy, can you create a quick visualization to show the top 10 most common terms in the descriptions?

Could you also share some of the less common terms that appear in multiple book descriptions?

Thanks!
Lexi

Reply Forward

Key Objectives

1. Vectorize the cleaned and normalized text using Count Vectorizer with the default parameters
2. Modify the Count Vectorizer parameters to reduce the number of columns:
    a) Remove stop words
    b) Set a minimum document frequency of 10%
3. Use the updated Count Vectorizer to identify the:
    a) Top 10 most common terms
    b) Top 10 least common terms that appear in at least 10% of the documents
4. Create a horizontal bar chart of the top 10 most common terms

TF-IDF

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Term Frequency-Inverse Document Frequency (TF-IDF) is an alternative to the word count calculation in a DTM

- It emphasizes important words by reducing the impact of common words

Term Frequency

Problem it solves:

High counts can dominate, especially for high frequency words or long documents

Solution:

Normalize the counts so they're all on the same scale

Inverse Document Frequency

Problem it solves:

Each word is treated equally, even when some might be more important

Solution:

Assign more weight to rare words than to common words

$$TF\text{-}IDF = \frac{Term\ count\ in\ document}{Total\ terms\ in\ document} * log\left(\frac{Total\ documents + 1}{Documents\ with\ the\ term + 1}\right)$$

TF-IDF VECTORIZER IN PYTHON

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Create a **TF-IDF Vectorizer** object in Python to use TD-IDF scores in your DTM

- It has many of the same parameters as the Count Vectorizer

```
# basic tfidf vectorizer code
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer()
tfidf = tv.fit_transform(text_clean)
tfidf_df = pd.DataFrame(tfidf.toarray(), columns=tv.get_feature_names_out())
tfidf_df
```

	dozen	eureka	favorite	gallon	half	ice	lemon	lemonade	life	market	maven	palmer	sale	tea	today
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.375318	0.543168	0.75107	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.411442	0.000000	0.00000	0.595449	0.690041	0.000000	0.000000	0.000000	0.000000
2	0.600547	0.000000	0.000000	0.600547	0.000000	0.000000	0.300100	0.434311	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.300100	0.000000	0.00000	0.434311	0.000000	0.000000	0.600547	0.000000	0.600547
5	0.000000	0.556913	0.000000	0.000000	0.000000	0.000000	0.556591	0.000000	0.00000	0.402755	0.466736	0.000000	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000	0.504577	0.422875	0.000000	0.364907	0.00000	0.000000	0.000000	0.504577	0.000000	0.422875	0.000000
7	0.000000	0.000000	0.644859	0.000000	0.000000	0.540443	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.540443	0.000000

With the default parameters, we get the same 15 terms (columns) across 8 documents (rows), but with TF-IDF scores instead of word counts

TF-IDF VECTORIZER IN PYTHON

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Create a **TF-IDF Vectorizer** object in Python to use TD-IDF scores in your DTM

- It has many of the same parameters as the Count Vectorizer

```
# tfidf vectorizer with some parameter tweaks
tv2 = TfidfVectorizer(stop_words='english', ngram_range=(1,2), min_df=2)
tfidf2 = tv2.fit_transform(text_clean)
tfidf_df2 = pd.DataFrame(tfidf2.toarray(), columns=tv2.get_feature_names_out())
tfidf_df2
```

	ice	ice tea	lemon	lemon lemon	lemonade	market	maven	maven market	tea
0	0.000000	0.000000	0.568471	0.000000	0.822704	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.338644	0.000000	0.000000	0.490093	0.567948	0.567948	0.000000
2	0.000000	0.000000	0.568471	0.000000	0.822704	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.581897	0.813262	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.568471	0.000000	0.000000	0.822704	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.524634	0.439939	0.000000	0.379631	0.439939	0.439939	0.000000
6	0.516768	0.516768	0.000000	0.000000	0.445928	0.000000	0.000000	0.000000	0.516768
7	0.577350	0.577350	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.577350

With the same parameters as earlier, we're back down to 9 terms instead of 15

COUNTS VS TF-IDF SCORES

NLP Pipeline

Text Preprocessing with Pandas

Text Preprocessing with spaCy

Vectorization

Here's a comparison between **word counts & TF-IDF scores** from the same data:

	ice	lemon	lemonade	market	maven	tea
0	0	1	1	0	0	0
1	0	1	0	1	1	0
2	0	1	1	0	0	0
3	0	6	0	0	0	0
4	0	1	0	1	0	0
5	0	2	0	1	1	0
6	1	0	1	0	0	1
7	1	0	0	0	0	1

1. *"lemon", "market", and "maven" are all equal*
2. *The value of 6 for "lemon" skews the results*
3. *"lemonade" shows up three times and "tea" twice*

	ice	lemon	lemonade	market	maven	tea
0	0.000000	0.568471	0.822704	0.000000	0.000000	0.000000
1	0.000000	0.411442	0.000000	0.595449	0.690041	0.000000
2	0.000000	0.568471	0.822704	0.000000	0.000000	0.000000
3	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.568471	0.000000	0.822704	0.000000	0.000000
5	0.000000	0.670130	0.000000	0.484914	0.561947	0.000000
6	0.603613	0.000000	0.520868	0.000000	0.000000	0.603613
7	0.707107	0.000000	0.000000	0.000000	0.000000	0.707107

1. *"maven" and "market" have higher values since they are more rare*
2. *Everything ranges between 0 and 1*
3. *"lemonade" is high for rows 0 and 2, but lower than "tea" in row 6*

ASSIGNMENT: TF-IDF VECTORIZER

NEW MESSAGE

May 21, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **RE: Vectorization request**

Hi again – Can you do the same analysis as last time, but using TF-IDF instead and compare the two results? Thanks!

Hello,

Now that you've cleaned and normalized the book descriptions using pandas and spaCy, can you create a quick visualization to show the top 10 most common terms in the descriptions?

Thanks!
Lexi

Reply Forward

Key Objectives

1. Vectorize the cleaned and normalized text using TF-IDF Vectorizer with the default parameters
2. Modify the TF-IDF Vectorizer parameters to reduce the number of columns:
    a) Remove stop words
    b) Set a minimum document frequency of 10%
    c) Set a maximum document frequency of 50%
3. Using the updated TF-IDF Vectorizer, create a horizontal bar chart of the top 10 most highly weighted terms
4. Compare the Count Vectorizer bar chart from the previous assignment with the TF-IDF Vectorizer bar chart and note the differences in the top term lists

KEY TAKEAWAYS

NLP projects have an extra **text preprocessing** step in the DS workflow

- *Text preprocessing is a part of the NLP pipeline, which is the series of steps your text data goes through for processing and analysis, including gathering, cleaning (general and text-specific), exploring and modeling*

Text cleaning & normalization can be done using Pandas and spaCy

- *Pandas is good for simple tasks like lowercasing and removing text with regular expressions*
- *spaCy can perform more advanced linguistic tasks like tokenization, lemmatization, removing stop words, and more*
- *By putting the steps into Python functions, you can better organize your code and create an NLP pipeline*

Vectorization is the process of making text numeric for future analysis

- *Vectorization starts by creating a document-term matrix and often follows the bag of words model*
- *The values can be filled with term counts (Count Vectorizer) or TF-IDF scores (TF-IDF Vectorizer)*
- *Later we'll talk about embeddings, which is another technique that takes word order and meaning into account*

NLP WITH MACHINE LEARNING

NLP WITH MACHINE LEARNING

In this section, we'll highlight tasks that can be solved using **traditional NLP methods**, including rules-based, and supervised & unsupervised **machine learning** techniques

TOPICS WE'LL COVER:

- Machine Learning Refresher
- Traditional NLP Overview
- Sentiment Analysis
- Text Classification
- Topic Modeling

GOALS FOR THIS SECTION:

- Understand how rules-based sentiment analysis techniques work, and become familiar with the VADER library
- Use Naïve Bayes and Logistic Regression as supervised learning approaches for text classification the with scikit-learn library
- Use Non-Negative Matrix Factorization (NMF) as an unsupervised learning approach for topic modeling with the scikit-learn library

WHAT IS MACHINE LEARNING?

Data scientists use **machine learning** algorithms to enable computers to learn and make decisions from data

Machine learning algorithms fall into two broad categories:

Supervised Learning

Using historical data to predict the future

What will house prices look like for the next 12 months?

How can I flag suspicious emails as spam?

Unsupervised Learning

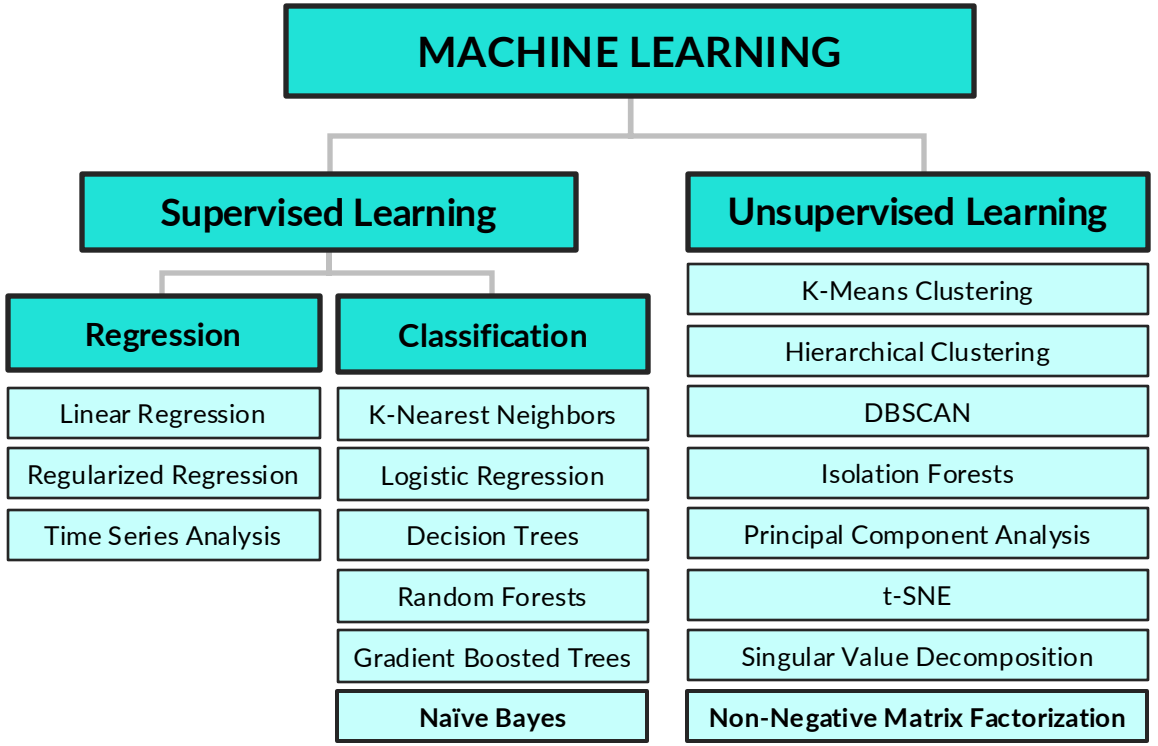
Finding patterns and relationships in data

How can I segment my customers?

What hidden themes are in these product reviews?

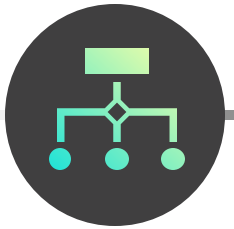
COMMON ML ALGORITHMS

You can use any of these **common machine learning algorithms** for natural language processing tasks once you've preprocessed your text data:



*In this section, we'll introduce these two techniques that are **commonly used for NLP tasks** like text classification (Naïve Bayes) and topic modeling (NMF)*

**The majority of these algorithms are explained in detail in courses 2-4 of this Data Science in Python series (Regression, Classification and Unsupervised Learning)*



TRADITIONAL NLP OVERVIEW

- Machine Learning Refresher
- Traditional NLP Overview
- Sentiment Analysis
- Text Classification
- Topic Modeling

These common NLP tasks are often solved using **traditional NLP methods**, such as simple rules-based techniques or more advanced ML algorithms

Sentiment Analysis

Identifying the positivity or negativity of text

- **Technique:** Rules-based
- **Library:** VADER
- **Input format:** raw text

Text Classification

Classifying text as one label or another

- **Technique:** Supervised learning (i.e., Naïve Bayes)
- **Library:** scikit-learn
- **Input format:** CV / TF-IDF

Topic Modeling

Finding themes within a corpus of text

- **Technique:** Unsupervised learning (i.e., NMF)
- **Library:** scikit-learn
- **Input format:** CV / TF-IDF

TRADITIONAL VS. MODERN NLP

When should I use traditional vs. modern NLP techniques?

- In summary, start simple!

What is my NLP goal?

Sentiment Analysis

Text Classification

Topic Modeling

→ *These can be done with traditional techniques* →

How much data do I have?

Small to medium data (<100k rows) → ***Try traditional techniques first***

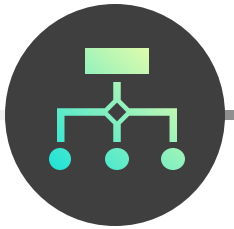
Big data (>1M rows) → ***Consider modern techniques***

Text Generation

Machine Translation

Question Answering

→ *These cannot be done with traditional techniques, so **use modern techniques***



SENTIMENT ANALYSIS

- Machine Learning Refresher
- Traditional NLP Overview
- Sentiment Analysis
- Text Classification
- Topic Modeling

Sentiment analysis is used to determine the positivity or negativity of text

- An overall sentiment score between -1 and +1 is given to each block of text

"A dozen lemons will make a gallon of lemonade." — **Neutral**

"When life gives you lemons, make lemonade! ☺*"* — **Positive**

"I didn't like the taste of that lemonade at all." — **Negative**

These are hints that this is positive / negative text

You'll notice that **sentiment analysis is applied on raw text** – it's not cleaned because punctuation matters, and it's not vectorized because word order matters

SENTIMENT ANALYSIS IN PYTHON

- Machine Learning Refresher
- Traditional NLP Overview
- Sentiment Analysis
- Text Classification
- Topic Modeling

Sentiment analysis can be done using rules-based techniques with libraries like VADER, classification techniques *(up next)*, or modern NLP techniques *(later)*

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

text = 'When life gives you lemons, make lemonade! 🙂'

analyzer = SentimentIntensityAnalyzer()
analyzer.polarity_scores(text)
```

```
{'neg': 0.0, 'neu': 0.75, 'pos': 0.25, 'compound': 0.4587}
```

0% of the text is negative, 75% is neutral, and 25% is positive

Overall sentiment score is positive!

VADER assigns predefined sentiment weights to words (amazing = 2.8, horrible = -2.5), incorporates modifiers (not, very, caps, punctuation, etc.), and computes a final score

**The entire 7000+ word lexicon can be found here: https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/vader_lexicon.txt*

ASSIGNMENT: SENTIMENT ANALYSIS

NEW MESSAGE

May 22, 2025

From: **Oscar Wynn** *(The Movie Maven)*

Subject: **Feel good vs dark movies**

Hi there,

We're a small entertainment news and movie reviews website, focused on data-driven content.

We're publishing an article on the top 10 most feel-good movies and the top 10 darkest movies according to data.

Could you use sentiment analysis to help us come up with movies for these two lists?

Thanks!
Oscar

movie_reviews.csv

Reply | Forward

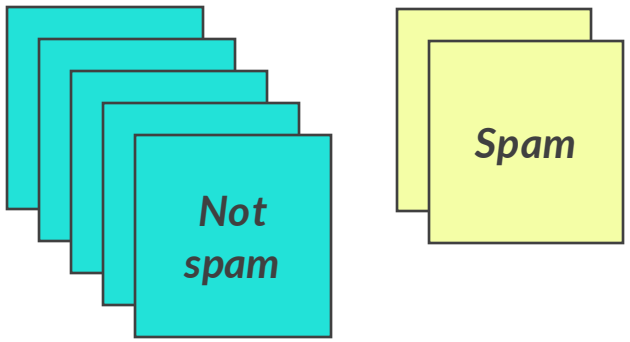
Key Objectives

1. Create a new "nlp_machine_learning" environment
2. Launch Jupyter Notebook
3. Read in the movie_reviews.csv file
4. Apply sentiment analysis to the movie_info column
5. Sort the sentiment scores to return the top 10 and bottom 10 sentiment scores and their corresponding movie titles

TEXT CLASSIFICATION

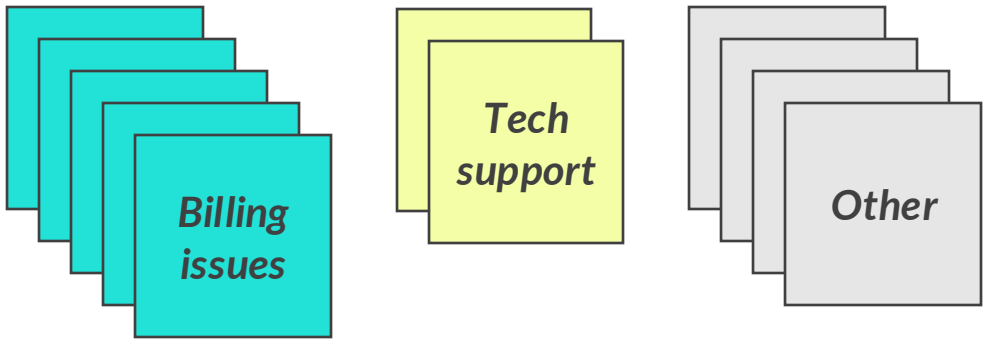
- Machine Learning Refresher
- Traditional NLP Overview
- Sentiment Analysis
- Text Classification
- Topic Modeling

Text classification is used to categorize text into groups based on *labeled* data



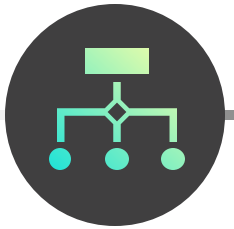
These existing emails have been prelabeled as spam or not spam

Send money ASAP! ← *Given this new email, text classification will tell us if it's spam or not spam*



These existing customer support tickets have been prelabeled as billing issues, tech support and other

Help me reset my password ← *Given this new ticket, text classification will tell us what type of ticket it is*



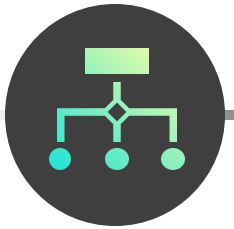
TEXT CLASSIFICATION ALGORITHMS

You can input vectorized text data into any **classification algorithm**:

- KNN, Logistic Regression, Decision Trees, Random Forests, Gradient Boosted Trees, etc.
- Naïve Bayes is another classification algorithm that works especially well on text data

Which classification algorithm should I choose for my text data?

- For small data sets *(<10k rows)*, start with **Naïve Bayes** and other simple models like Logistic Regression, KNN, etc.
- For medium data sets *(<100k rows)*, start with **Logistic Regression** and other classification techniques like Decision Trees, Random Forests, Gradient Boosted Trees, etc.
- For large data sets *(>1M rows)*, start with **Gradient Boosted Trees** and potentially move on to modern NLP techniques with LLMs



NAÏVE BAYES

Machine Learning Refresher

Traditional NLP Overview

Sentiment Analysis

Text Classification

Topic Modeling

Naïve Bayes is a technique that's commonly used for text classification

- It's based on Bayes Theorem, which assumes conditionally independent features
- This independence assumption is naïve, but the algorithm works surprisingly well on text data

EXAMPLE *If an email contains the word "ASAP", how likely is it going to be spam?*

The probability that the word ASAP appears in a spam email

The probability an email is spam

$$P(Spam|ASAP) = \frac{P(ASAP|Spam) * P(Spam)}{P(ASAP)}$$

*The probability that an email is **spam**, given it contains the word **ASAP***

The probability the word ASAP is in any email

NAÏVE BAYES

Naïve Bayes is a technique that's commonly used for text classification

- It's based on Bayes Theorem, which assumes conditionally independent features
- This independence assumption is naïve, but the algorithm works surprisingly well on text data

EXAMPLE *If an email contains the word "ASAP", how likely is it going to be spam?*

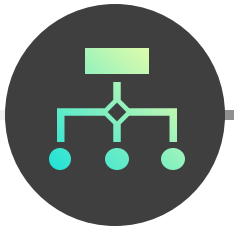
Distribution of 1,000 emails:

ASAP \ SPAM	1	0
1	50	10
0	200	740

$$\frac{P(ASAP|Spam) * P(Spam)}{P(ASAP)} = \frac{\frac{50}{250} * \frac{250}{1000}}{\frac{60}{1000}} = \frac{0.2 * 0.25}{0.06} \approx 0.83$$

There's an 83% chance an email is spam if it contains the word "ASAP"

When looking at one word, the calculation is just **Bayes Theorem**



NAÏVE BAYES

Machine Learning Refresher

Traditional NLP Overview

Sentiment Analysis

Text Classification

Topic Modeling

Naïve Bayes is a technique that's commonly used for text classification

- It's based on Bayes Theorem, which assumes conditionally independent features
- This independence assumption is naïve, but the algorithm works surprisingly well on text data

EXAMPLE *If an email contains the word "ASAP" and the "$" symbol, how likely is it going to be spam?*

The probability that the word ASAP appears in a spam email

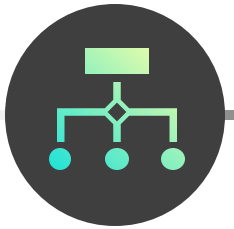
The probability that $ appears in a spam email

The probability an email is spam

$$P(Spam|ASAP,\$) = \frac{P(ASAP|Spam) * P(\$|Spam) * P(Spam)}{P(ASAP,\$)}$$

*The probability that an email is **spam**, given it contains **ASAP** and **$***

The probability that an email contains both ASAP and $



NAÏVE BAYES

Naïve Bayes is a technique that's commonly used for text classification

- It's based on Bayes Theorem, which assumes conditionally independent features
- This independence assumption is naïve, but the algorithm works surprisingly well on text data

EXAMPLE *If an email contains the word "ASAP" and the "$" symbol, how likely is it going to be spam?*

This is the naïve assumption – the probability that an email contains ASAP and the probability it contains $ are not independent, they're actually correlated

The probability an email is spam

$$P(Spam|ASAP,\$) = \frac{P(ASAP|Spam) * P(\$|Spam) * P(Spam)}{P(ASAP,\$)}$$

*The probability that an email is **spam**, given it contains **ASAP** and **$***

The probability that an email contains both ASAP and $

NAÏVE BAYES

Naïve Bayes is a technique that's commonly used for text classification

- It's based on Bayes Theorem, which assumes conditionally independent features
- This independence assumption is naïve, but the algorithm works surprisingly well on text data

EXAMPLE *If an email contains the word "ASAP" and the "$" symbol, how likely is it going to be spam?*

Distribution of 1,000 emails:

ASAP \ SPAM	1	0
1	50	10
0	200	740

$ \ SPAM	1	0
1	80	30
0	170	720

ASAP and $: 17

$$\frac{P(ASAP|Spam) * P(\$|Spam) * P(Spam)}{P(ASAP, \$)} \approx 0.94$$

These probabilities are all calculated automatically by the model!

There's a 94% chance an email is spam if it contains "ASAP" and "$"

NAÏVE BAYES IN PYTHON

Use sklearn's **MultinomialNB** to perform Naïve Bayes in Python

- The input should be a CountVectorizer or TfidfVectorizer
- There are no parameters to tune with Naïve Bayes

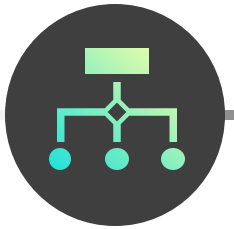
```
from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
```

This follows the typical sklearn process for a supervised learning model:*

1. *Instantiate an object*
2. *Fit a model*
3. *Make a prediction*

We're using MultinomialNB because **the inputs are counts** (like you would see in a CountVectorizer output) – for 1/0 values, like in the previous spam example, you would use BernoulliNB instead

**This sklearn process is discussed in much more detail in the Classification course in the Data Science in Python series*



TEXT CLASSIFICATION NEXT STEPS

Once you fit your first Naïve Bayes model in Python, you can improve your text classification model by tuning any part of the NLP pipeline:

1 Text preprocessing

- Update any cleaning or normalization steps

2 Vectorization

- Fine tune the CountVectorizer parameters *(stop_words, ngram_range, min_df, etc.)*
- Try using TfidfVectorizer instead

3 Feature engineering

- Include non-term values such as text length, sentiment score, time of day sent, etc.

4 Modeling

- Try a different probability cutoff point instead of the default 50% probability
- Try a different classification model *(Logistic Regression, Gradient Boosted Trees, etc.)*

ASSIGNMENT: TEXT CLASSIFICATION

NEW MESSAGE

May 23, 2025

From: **Oscar Wynn** *(The Movie Maven)*

Subject: **Female vs male directors**

Hi again,

Our next piece is going to spotlight female directors, and we want to see if there are any differences between the types of movies that female versus male directors create.

Could you create a classification model that predicts which movies are directed by females versus males based their movie descriptions?

Please also send over a list of the top 5 movies that are most likely directed by a female according to the model.

Thanks!

Reply Forward

Key Objectives

1. Clean and normalize the "movie_info" column using the "maven_text_preprocessing.py" module
2. Create a Count Vectorizer
    - Remove stop words
    - Set the minimum document frequency to 10%
3. Create a Naïve Bayes model and a Logistic Regression model to predict which movies are directed by women vs men using the CV
4. Compare their accuracy scores and classification reports
5. Using the better performing model, return the top 5 movies that the model predicts are most likely directed by a women

TOPIC MODELING

Topic modeling is used to find themes in *unlabeled* text documents

- Topic modeling techniques extract the topics, but it's up to you to interpret and name them

	Topic 1	*Topic 2*
"I like lemons and limes."	100%	0%
"Puppies and kittens are so cute."	0%	100%
"I'm making cat-shaped cookies."	50%	50%
"My dog loves apples and blueberries."	67%	33%

What are topics 1 and 2?

- Topic 1: lemons, limes, cookies, apples, blueberries — **Food**
- Topic 2: puppies, kittens, cat, dog — **Animals**

TOPIC MODELING ALGORITHMS

You can input vectorized text data into a **topic modeling algorithm**

Which topic modeling algorithm should I choose for my text data?

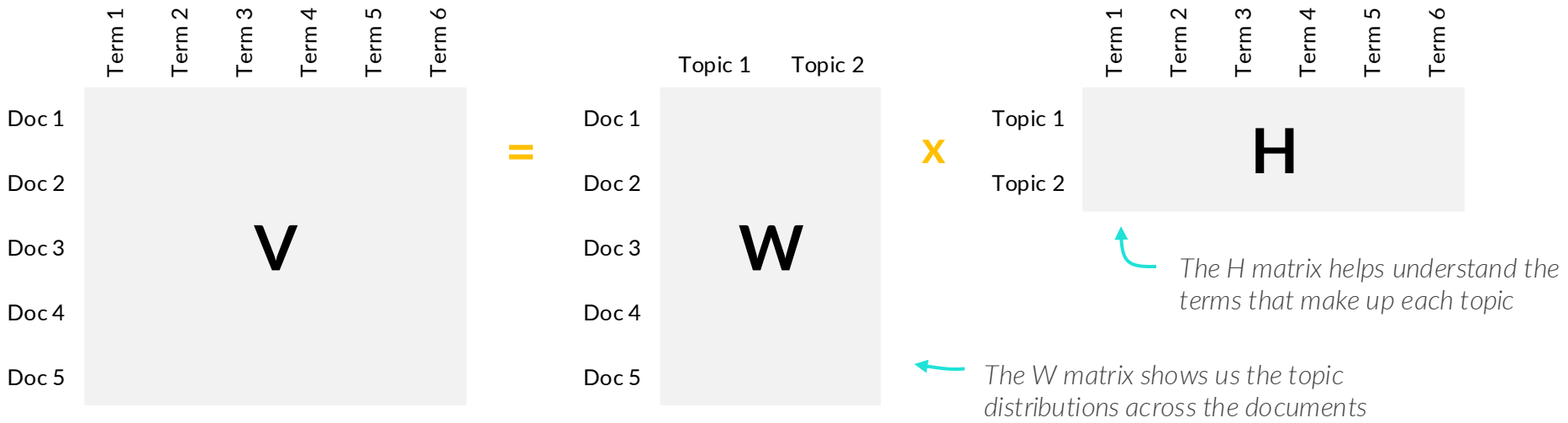
- For small data sets *(<10k rows)*, start with **Non-Negative Matrix Factorization (NMF)** using the sklearn library
- For medium data sets *(<100k rows)*, start with **Latent Dirichlet Allocation (LDA)** using the genism library
- For large data sets *(>1M rows)*, use modern embedding-based NLP approaches such as **BERTopic** and **Top2Vec** (*embeddings will be discussed later!)*

In this course, we'll be demoing **NMF** because it's in sklearn. For more details on LDA, you can check out my YouTube video on LDA using genism: https://www.youtube.com/watch?v=NYkbqzTlW3w

NON-NEGATIVE MATRIX FACTORIZATION

Non-Negative Matrix Factorization (NMF) is a topic modeling technique that decomposes the document-term matrix (V) into two other matrices:

- A document-topic matrix (W) that shows how much each topic appears in each document
- A topic-term matrix (H) that shows how important each word is to each topic



Other matrix factorization techniques include PCA and SVD, but **NMF is the only one that returns all positive results**, which is needed for text data where negative values wouldn't make sense

NMF IN PYTHON

Machine Learning Refresher

Traditional NLP Overview

Sentiment Analysis

Text Classification

Topic Modeling

Use sklearn's **NMF** from the decomposition module to perform NMF in Python

- The input should be a CountVectorizer or TfidfVectorizer
- Start at 2 components (topics) and increase by 1 until you figure out the best number of topics

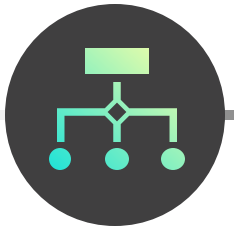
```
from sklearn.decomposition import NMF

nmf_model = NMF(n_components=2, random_state=42)
W = nmf_model.fit_transform(X)
H = nmf_model.components_
```

This follows the typical sklearn process for an unsupervised learning model:*

1. *Instantiate an object*
2. *Fit and transform the data*
3. *View the attributes*

NMF starts with an initial set of randomized values, so set a random state to get the same results each time

**This sklearn process is discussed in much more detail in the Unsupervised Learning course in the Data Science in Python series*



TOPIC MODELING NEXT STEPS

Once you fit your first NMF model in Python, you can improve your topic model by tuning any part of the NLP pipeline:

1 Text preprocessing

- Update any cleaning or normalization steps

2 Vectorization

- Fine tune the TfidfVectorizer parameters *(stop_words, ngram_range, min_df, etc.)*
- Try using CountVectorizer instead

3 Modeling

- Modify "n_components" to try out different numbers of topics
- Try a different topic modeling technique *(Latent Dirichlet Allocation, Latent Semantic Analysis, BERTopic, Top2Vec, etc.)*

BONUS: In the demo, we'll show an example of how you can mix and match multiple algorithms for your analysis (in this case, topics + sentiment scores + EDA = sentiment about each topic)

ASSIGNMENT: TOPIC MODELING

NEW MESSAGE

May 27, 2025

From: **Oscar Wynn** *(The Movie Maven)*

Subject: **Movie themes**

Hello,

Our feel-good movies list and female directors articles were both hits over the weekend! Thanks for your help with those.

Our next goal is to suggest movies based on movie themes. Could you use topic modeling to find the major themes in our movie list?

Once you do that, for a few of the themes, can you provide a list of the top 5 movies that have the theme?

Thanks!
Oscar

Reply Forward

Key Objectives

1. Using the same preprocessed data as the last assignment, create a Tfidf Vectorizer
    - Remove stop words
    - Start with min_df = 0.05 and max_df=0.2
2. Create an NMF model to find the main topics in the movie descriptions
    - Start with n_components=2
3. Tweak the model by updating the Tfidf Vectorizer parameters and number of topics
4. Interpret and name the topics
5. For two of the topics, return the top movies that contain the topic

KEY TAKEAWAYS

Machine learning techniques are a great starting point for NLP tasks

- *Any general ML algorithm can be applied for NLP tasks once the text data is cleaned and vectorized*
- *ML is the preferred approach for small & medium data sets, while modern NLP is preferred for large ones*

Sentiment analysis is often done using rules-based techniques

- *Popular libraries within Python for sentiment analysis include VADER and TextBlob*
- *Other techniques include text classification and modern NLP techniques*

Naïve Bayes is a popular **text classification** technique

- *Naïve Bayes is a good starting point for classifying text data within small data sets (< 10k rows)*
- *Other techniques include Logistic Regression, Gradient Boosted Trees, and modern NLP techniques*

Non-Negative Matrix Factorization is a popular **topic modeling** technique

- *NMF is a good starting point for identifying topics within small data sets (< 10k rows)*
- *Other techniques include Latent Dirichlet Allocation and modern NLP techniques*

NEURAL NETWORKS & DEEP LEARNING

NEURAL NETWORKS & DEEP LEARNING

In this section, we'll visually break down the concepts behind **neural networks** and **deep learning**, the building blocks of modern NLP techniques

TOPICS WE'LL COVER:

- Modern NLP Overview
- Neural Networks
- Deep Learning

GOALS FOR THIS SECTION:

- Understand how logistic regression works, and build up to a neural network step-by-step
- Become familiar with key terms such as layers, nodes, weights, activation functions, and more
- Learn how to create a neural network in Python with MLPClassifier & MLPRegressor
- Understand the difference between neural networks and deep learning
- Get introduced to deep learning architectures

MODERN NLP OVERVIEW

We are now moving from **traditional** to **modern NLP**:

- Modern NLP Overview
- Neural Networks
- Deep Learning

Traditional NLP

Data:

- Small to medium data sets

Techniques:

- Rules-based
- Supervised learning (Naïve Bayes)
- Unsupervised learning (NMF)

Applications:

- Sentiment analysis
- Text classification
- Topic modeling

Modern NLP

Data:

- Small to large data sets

Techniques:

- Transformers-based LLMs (BERT, GPT, LLaMA, T5, BART)

To understand transformer-based models, we'll start with the basics: neural networks

Applications:

- Traditional NLP applications
- Text summarization
- Text generation

MODERN NLP OVERVIEW

In the next two sections, we'll be covering these **modern NLP concepts** to understand how LLMs work before applying them using Hugging Face:

- Modern NLP Overview
- Neural Networks
- Deep Learning

COMPLEXITY

Concepts

1 Neural Networks & Deep Learning

a) Logistic Regression
b) Neural Networks
c) Deep Learning

2 Transformers & LLMs

a) Embeddings
b) Attention
c) Transformer-Based LLMs

Key Terms

- *Neural network components: layers, nodes, weights, parameters, activation functions*
- *Neural network training: forward pass, loss, backpropagation, gradient descent*
- *Deep learning architectures: FNN, CNN, RNN, LSTM, Transformers*
- *Embeddings: tokens*
- *Attention: queries, keys, scores*
- *Feedforward neural network*
- *Transformers: encoders vs decoders*
- *Pretrained LLMs: BERT, GPT and more*

INTRO TO NEURAL NETWORKS

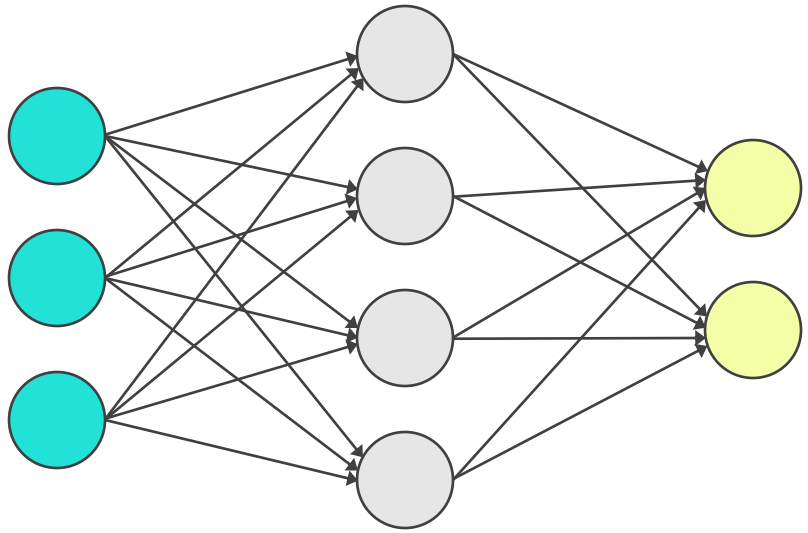
Modern NLP Overview

Neural Networks

Deep Learning

A **neural network** is a machine learning model designed to process information in a way that's inspired by neurons in the human brain

- Biological neurons communicate by receiving and passing along information to other neurons
- A neural network processes data through layers of nodes (neurons)



Input layer
(Features)

Hidden layer
(Parameters & Activation functions)

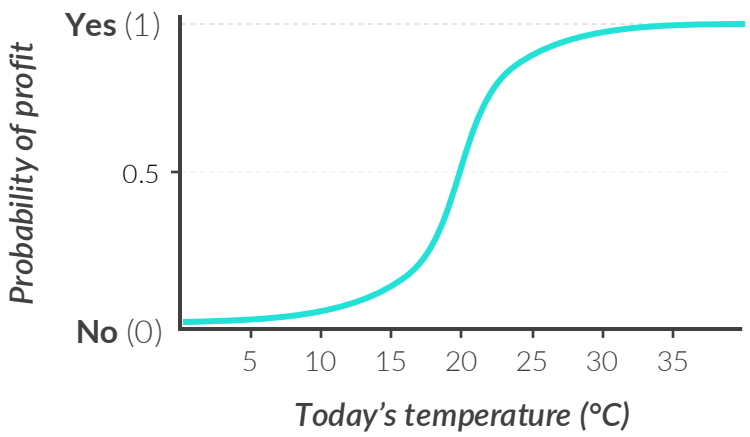
Output layer
(Predictions)

LOGISTIC REGRESSION

To understand neural networks, let's start with a simple **logistic regression** model

- Logistic regression is a classification technique used to predict a true or false outcome

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



The higher the temperature today, the more likely my lemonade stand will be profitable

Probability of profit *Today's temperature*

$$p = \sigma(mx + b)$$

x = 15 (59F) → *p = 22%*
x = 25 (77F) → *p = 78%*
x = 35 (95F) → *p = 98%*

What are **σ**, **m**, and **b**?

- *Coming up next…*

LOGISTIC REGRESSION

Modern NLP Overview

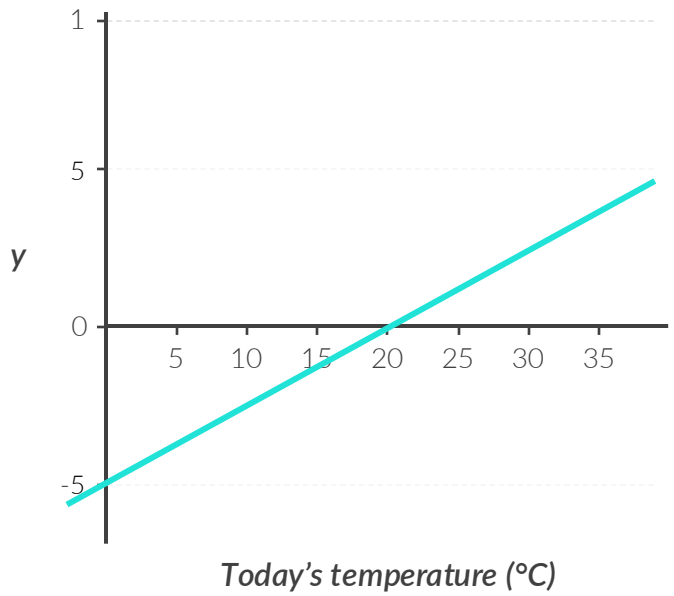
Neural Networks

Deep Learning

To understand neural networks, let's start with a simple **logistic regression** model

- Logistic regression is a classification technique used to predict a true or false outcome

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



Slope *Intercept*

$$y = mx + b = 0.25x - 5$$

x = 15 (59F) → *y = -1.25*
x = 25 (77F) → *y = 1.25*
x = 35 (95F) → *y = 3.75*

How do we come up with values for **m** and **b**?

- *We fit a logistic regression model in scikit-learn*

LOGISTIC REGRESSION

Modern NLP Overview

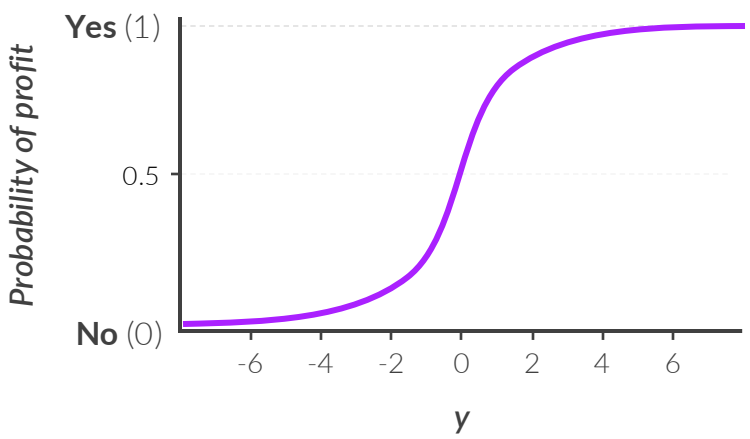
Neural Networks

Deep Learning

To understand neural networks, let's start with a simple **logistic regression** model

- Logistic regression is a classification technique used to predict a true or false outcome

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



The sigmoid function transforms the y-values so they fall between 0 and 1

$$p = \sigma(y)$$

x = 15 (59F) → *y = -1.25* → *p = 22%*

x = 25 (77F) → *y = 1.25* → *p = 78%*

x = 35 (95F) → *y = 3.75* → *p = 98%*

These probability values are much more interpretable than the original y-values

LOGISTIC REGRESSION

Modern NLP Overview

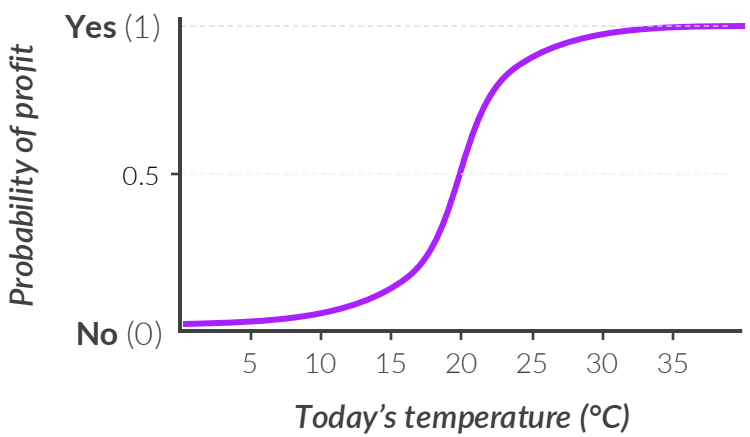
Neural Networks

Deep Learning

To understand neural networks, let's start with a simple **logistic regression** model

- Logistic regression is a classification technique used to predict a true or false outcome

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



$$p = \frac{1}{1 + e^{-(mx+b)}}$$

Remember, y=mx+b

This is the calculation for a sigmoid (σ) transformation

LOGISTIC REGRESSION

Modern NLP Overview

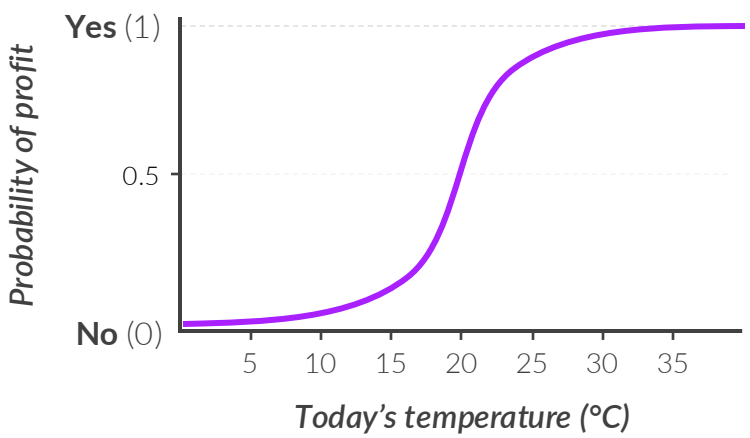
Neural Networks

Deep Learning

To understand neural networks, let's start with a simple **logistic regression** model

- Logistic regression is a classification technique used to predict a true or false outcome

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



2. Non-linear transformation

1. Linear transformation

$$p = \sigma(mx + b)$$

x = 15 (59F) → *y = -1.25* → *p = 22%*

x = 25 (77F) → *y = 1.25* → *p = 78%*

x = 35 (95F) → *y = 3.75* → *p = 98%*

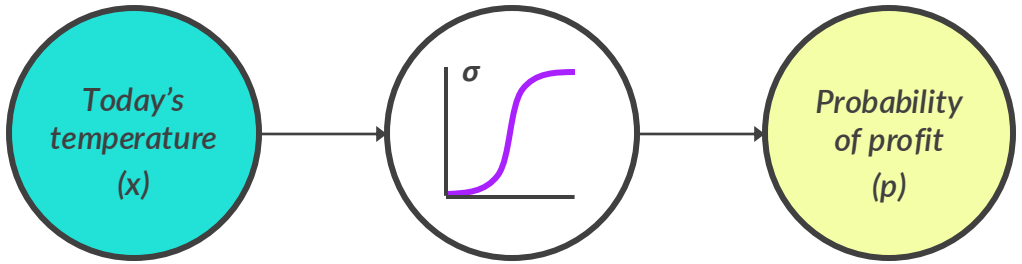
Why are these steps so important?

- *A linear transformation followed by a non-linear transformation is the main calculation of a **neural network** (coming up next!)*

LOGISTIC REGRESSION: VISUALLY

A **logistic regression** model is essentially a very simple **neural network**

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



Input layer
(Features)

Hidden layer
(Parameters & Activation functions)

Output layer
(Predictions)

*This sigmoid function is a type of non-linear transformation, or in NN-speak, an **activation function***

$$p = \sigma(wx + b)$$

*In a neural network, the slope (m) is called a **weight (w),** the intercept (b) is called a **bias (b)**, and together, they are called the model **parameters***

Let's include more inputs, profitability can't just depend on temperature!

LOGISTIC REGRESSION: VISUALLY

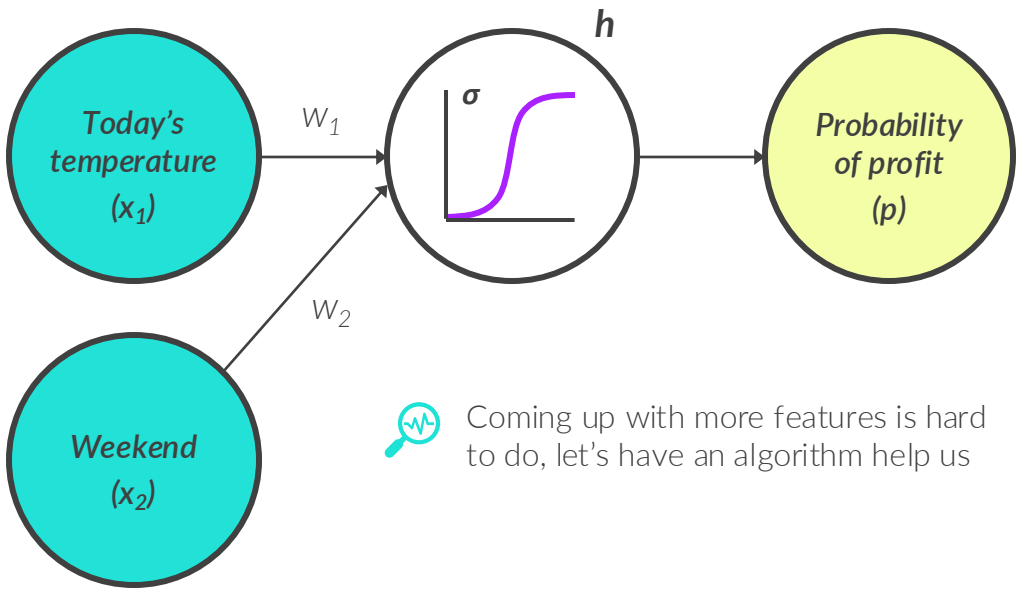
Modern NLP Overview

Neural Networks

Deep Learning

A **logistic regression** model is essentially a very simple **neural network**

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



Coming up with more features is hard to do, let's have an algorithm help us

*These **w** and **b** values come from fitting a logistic regression model in scikit-learn*

$$h = \sigma(w_1 x_1 + w_2 x_2 + b)$$

$$h = \sigma(0.25x_1 + 1x_2 - 5)$$

$x_1 = 15, x_2 = 0 \rightarrow h = 22\%$
$x_1 = 15, x_2 = 1 \rightarrow h = 43\%$
$x_1 = 25, x_2 = 0 \rightarrow h = 77\%$
$x_1 = 25, x_2 = 1 \rightarrow h = 90\%$
$x_1 = 35, x_2 = 0 \rightarrow h = 97\%$
$x_1 = 35, x_2 = 1 \rightarrow h = 99\%$

Input layer *(Features)*

Hidden layer *(Parameters & Activation functions)*

Output layer *(Predictions)*

NEURAL NETWORKS: VISUALLY

Modern NLP Overview

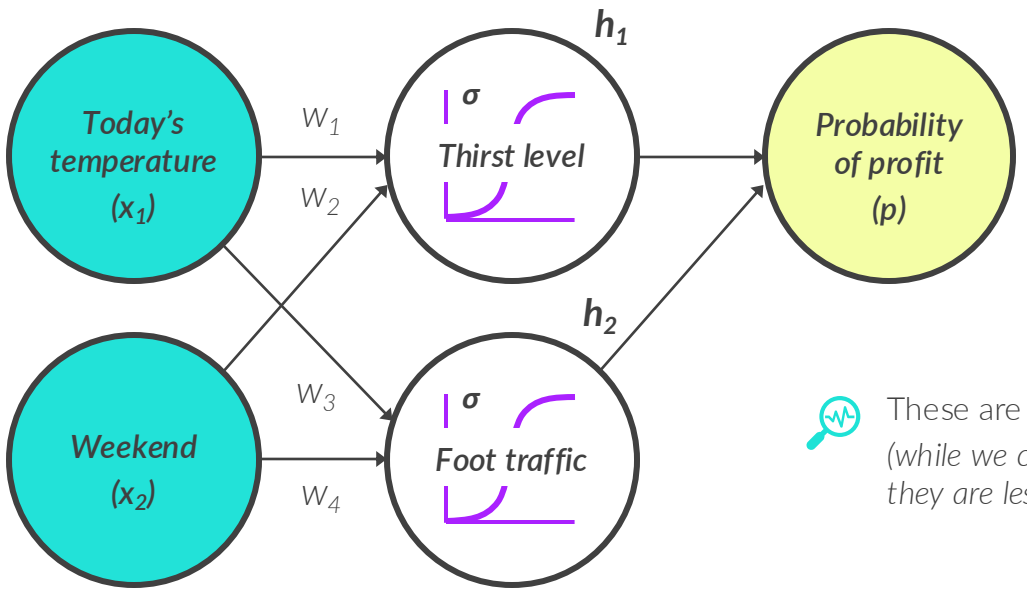
Neural Networks

Deep Learning

Adding nodes to the hidden layer makes this behave like a true **neural network**

- You can specify the number of nodes in the hidden layer and the activation function for each

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$
$$h_2 = \sigma(w_3 x_1 + w_4 x_2 + b_2)$$

These are features the model came up with!
(while we can interpret them here, in practice they are less interpretable)

Input layer
(Features)

Hidden layer
(Parameters & Activation functions)

Output layer
(Predictions)

NEURAL NETWORKS: VISUALLY

Modern NLP Overview

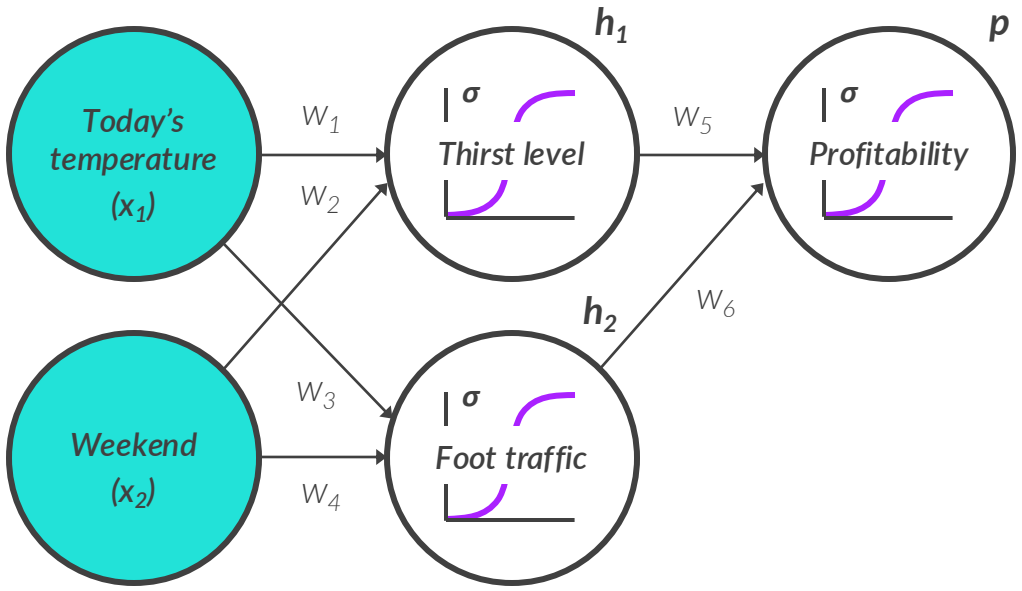
Neural Networks

Deep Learning

Adding nodes to the hidden layer makes this behave like a true **neural network**

- You can specify the number of nodes in the hidden layer and the activation function for each

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



Input layer
(Features)

Hidden layer
(Parameters & Activation functions)

Output layer
(Predictions)

$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$
$$h_2 = \sigma(w_3 x_1 + w_4 x_2 + b_2)$$
$$p = \sigma(w_5 h_1 + w_6 h_2 + b_3)$$

The outputs from the hidden layers are assigned their own weights and bias, and wrapped in a final activation function

NEURAL NETWORKS: VISUALLY

Modern NLP Overview

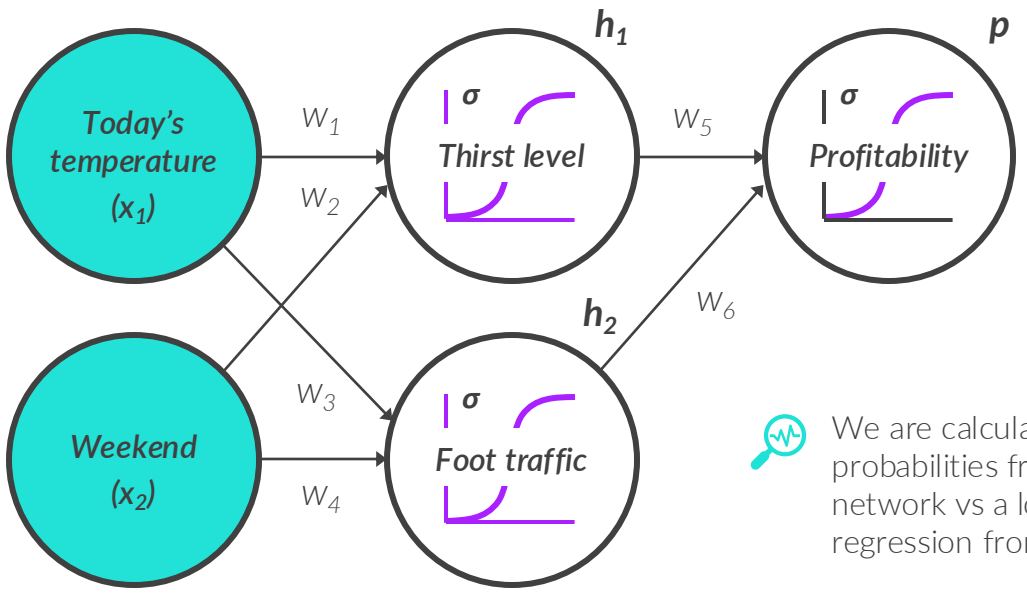
Neural Networks

Deep Learning

Adding nodes to the hidden layer makes this behave like a true **neural network**

- You can specify the number of nodes in the hidden layer and the activation function for each

EXAMPLE *Based on today's temperature, will my lemonade stand be profitable today?*



$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$
$$h_2 = \sigma(w_3 x_1 + w_4 x_2 + b_2)$$
$$p = \sigma(w_5 h_1 + w_6 h_2 + b_3)$$

We are calculating these probabilities from a neural network vs a logistic regression from earlier

$x_1 = 15, x_2 = 0 \rightarrow p = 15\%$
$x_1 = 15, x_2 = 1 \rightarrow p = 46\%$
$x_1 = 25, x_2 = 0 \rightarrow p = 73\%$
$x_1 = 25, x_2 = 1 \rightarrow p = 90\%$
$x_1 = 35, x_2 = 0 \rightarrow p = 91\%$
$x_1 = 35, x_2 = 1 \rightarrow p = 96\%$

Input layer
(Features)

Hidden layer
(Parameters & Activation functions)

Output layer
(Predictions)

NEURAL NETWORKS SUMMARY

Modern NLP Overview

Neural Networks

Deep Learning

1 A neural network processes data through layers of nodes

- There are input, hidden and output nodes
- It's up to you as a data scientist to decide on the number of hidden layers and nodes

2 The same two step calculation is done at every node

- Step 1: Calculate a ***weight***ed sum of the inputs & add a ***bias*** (*weights & bias* = ***parameters)***
- Step 2: Apply a non-linear transformation to the result, also called an ***activation function***

3 Neural networks are a type of supervised learning technique

- You input in historical data and labels, and get a prediction (*Python demo up next!*)
- Other supervised learning techniques include Logistic Regression, Naïve Bayes, etc.

4 How do you determine the parameters?

- In practice, you can fit a neural network using scikit-learn (*Python demo up next!)*
- In theory, the training process involves adjusting parameters (*we'll discuss very soon!)*

EXERCISE: NEURAL NETWORK COMPONENTS

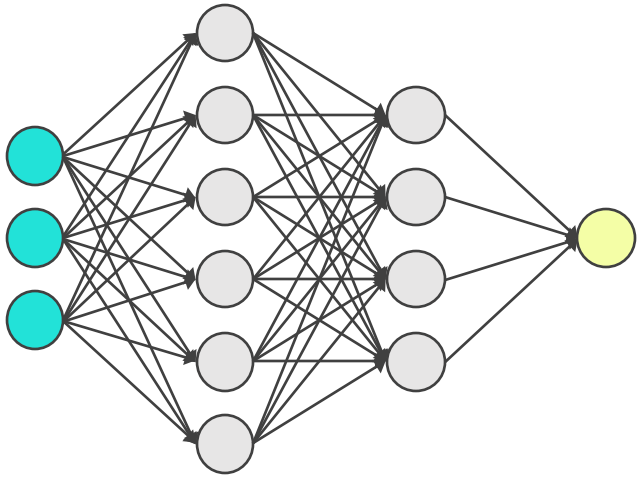
Modern NLP Overview

Neural Networks

Deep Learning

Drag these labels to the diagram or formula:

Input Layer	Hidden Layer	Output Layer	
Node	Weight	Bias	Activation Function



Answer these questions:

- *How many layers?*
- *How many hidden layers?*
- *How many nodes?*
- *How many weights?*
- *How many biases?*
- *How many parameters?*
- *How many activation functions?*

$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1)$$

NEURAL NETWORKS IN PYTHON

Modern NLP Overview

Neural Networks

Deep Learning

To create a neural network in Python, use **MLPClassifier** or **MLPRegressor** within sklearn's neural network module

- MLP stands for Multilayer Perceptron, which is another name for a neural network

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(hidden_layer_sizes=(100,), activation='relu')
```

Defines the number of nodes in each hidden layer

Examples:

- *(100,) – 1 hidden layer with 100 nodes (default)*
- *(50,30) – 2 hidden layers with 50 and 30 nodes respectively*

Sets the activation function for the hidden layers

Examples:

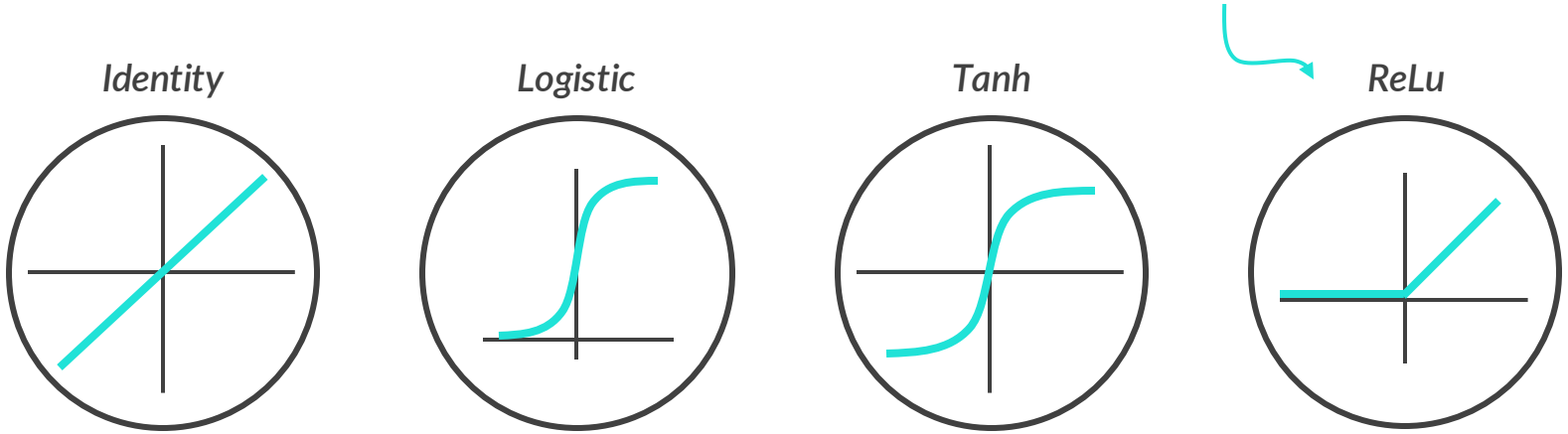
- *'relu' (default)*
- *'logistic'*
- *'tanh'*
- *'identity'*

NEURAL NETWORKS IN PYTHON

To create a neural network in Python, use **MLPClassifier** or **MLPRegressor** within sklearn's neural network module

- MLP stands for Multilayer Perceptron, which is another name for a neural network

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(hidden_layer_sizes=(100,), activation='relu')
```



NEURAL NETWORKS IN PYTHON

Modern NLP Overview

Neural Networks

Deep Learning

To create a neural network in Python, use **MLPClassifier** or **MLPRegressor** within sklearn's neural network module

- MLP stands for Multilayer Perceptron, which is another name for a neural network

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(hidden_layer_sizes=(100,),
                   activation='relu',
                   max_iter=200,
                   random_state=42)

nn.fit(X, y)
y_pred = nn.predict(X)
```

This follows the typical sklearn process for a supervised learning model:

1. *Instantiate an object*
2. *Fit a model*
3. *Make a prediction*

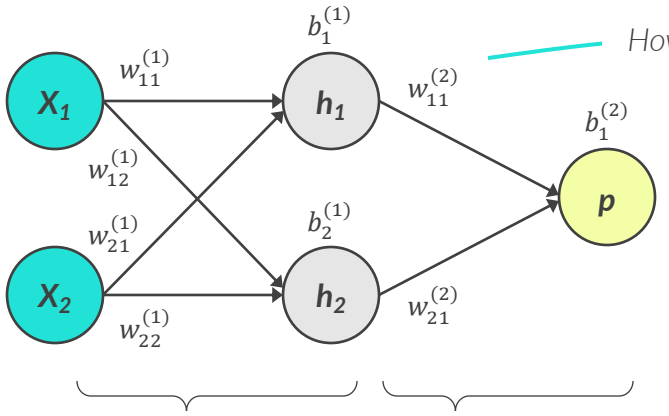
When are neural networks used in practice?

- While they could be used for classification and regression, it's often too much of a black box, and more interpretable models like Logistic Regression and Linear Regression are primarily used instead
- For modern NLP tasks, neural networks are rarely used on their own, but rather as building blocks for more complex techniques and architectures *(coming up soon!)*

PRO TIP: NN NOTATION & MATRICES

As we saw in the Python code, the weights and biases of a neural network are contained in **weight matrices** and **bias vectors**

- This is helpful to remember for the next section on Transformers & LLMs, where everything we review will live in matrices



How to read → $w_{11}^{(1)}$ ← *Weight matrix*

Starting node ↑ ↑ *Ending node*

Weight matrices:

$$\begin{bmatrix} 0.3 & 0.08 \\ 0.05 & 2.5 \end{bmatrix} \quad \begin{bmatrix} 3.5 \\ 3 \end{bmatrix}$$

Bias vectors:

$$\begin{bmatrix} -6.5 & -1.8 \end{bmatrix} \quad \begin{bmatrix} -3.2 \end{bmatrix}$$

$$h_1 = \sigma(0.3x_1 + 0.05x_2 - 6.5)$$
$$h_2 = \sigma(0.08x_1 + 2.5x_2 - 1.8)$$
$$p = \sigma(3.5h_1 + 3h_2 - 3.2)$$

TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

Training a neural network means to calculate its optimal parameters

1. **Random start**: Start with an initial set of random weights & biases
2. **Forward pass**: Starting from the left, apply all calculations through the neural network to get to a final set of predicted values
3. **Calculate loss**: Compare the predicted and actual values to compute the error, or loss
4. **Update parameters**: Starting from the right, calculate how much each parameter contributed to the loss with *back propagation*, and then use *gradient descent* (a popular optimization technique) to adjust the parameters by moving them a step closer to reducing the loss
5. **Repeat**: Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)

The math behind back propagation and gradient descent is beyond the scope of this course, but the key takeaway is that **each iteration moves closer to the optimal parameters**, and it does so as efficiently as possible

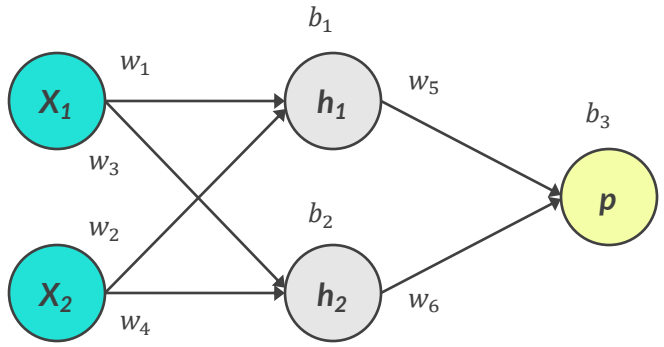
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

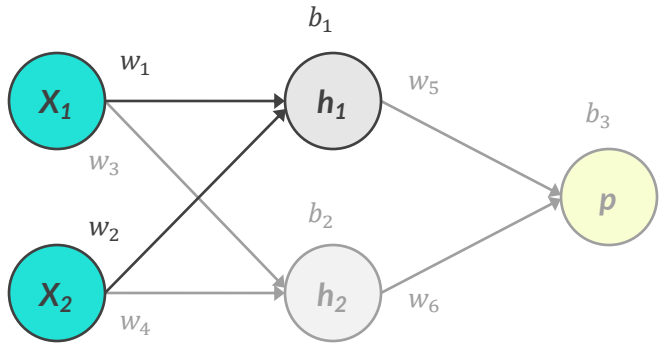
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$

Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

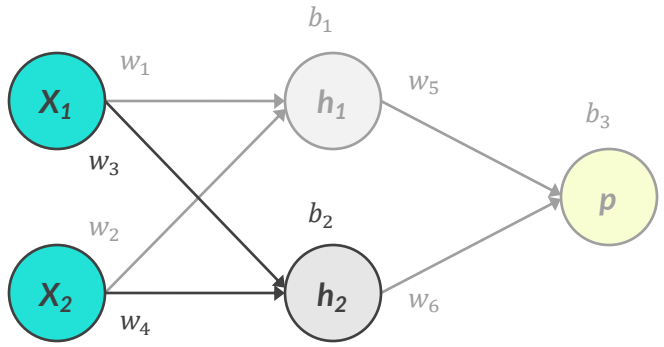
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$

$$h_2 = \sigma(w_3 x_1 + w_4 x_2 + b_2)$$

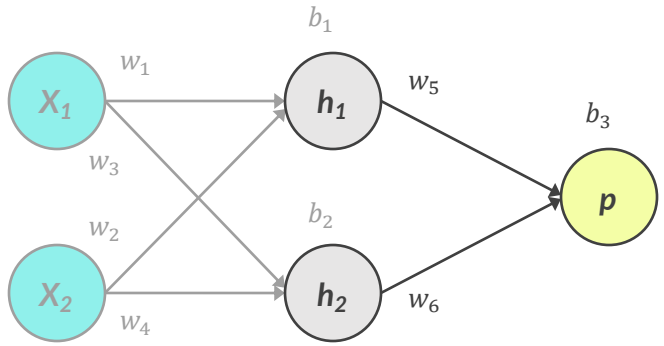
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$

$$h_2 = \sigma(w_3 x_1 + w_4 x_2 + b_2)$$

$$p = \sigma(w_5 h_1 + w_6 h_2 + b_3)$$

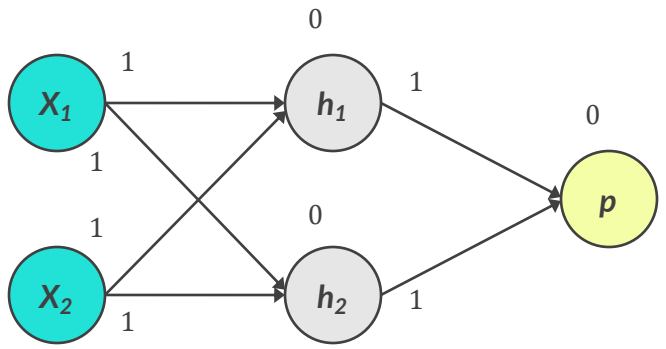
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

$$h_1 = \sigma(w_1 x_1 + w_2 x_2 + b_1)$$

$$h_2 = \sigma(w_3 x_1 + w_4 x_2 + b_2)$$

$$p = \sigma(w_5 h_1 + w_6 h_2 + b_3)$$

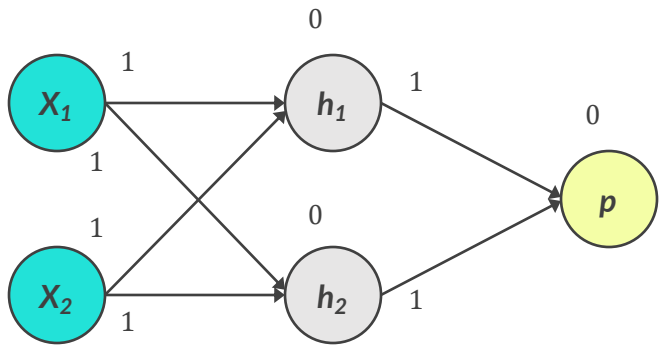
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

$$h_1 = \sigma(1x_1 + 1x_2 + 0)$$

$$h_2 = \sigma(1x_1 + 1x_2 + 0)$$

$$p = \sigma(1h_1 + 1h_2 + 0)$$

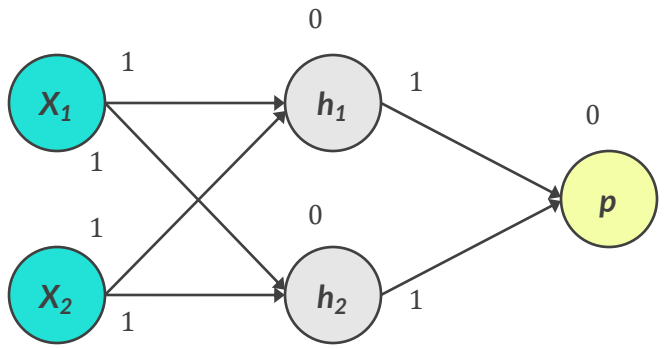
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 1: Random start – Start with an initial set of random weights & biases



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)
14	0	0
18	1	0
22	0	0
22	1	1
26	0	1
26	1	1
30	0	1
30	1	1
35	0	1
15	1	0

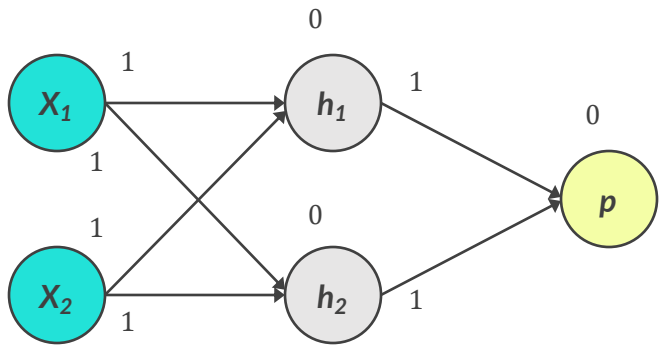
$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

TRAINING NEURAL NETWORKS

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)
14	0	0	
18	1	0	
22	0	0	
22	1	1	
26	0	1	
26	1	1	
30	0	1	
30	1	1	
35	0	1	
15	1	0	

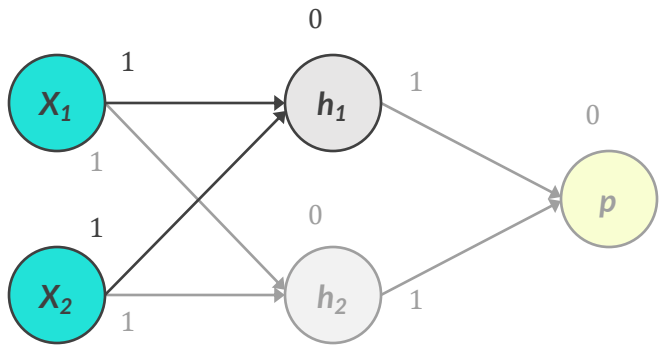
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)
14	**0**	**0**	
18	1	0	
22	0	0	
22	1	1	
26	0	1	
26	1	1	
30	0	1	
30	1	1	
35	0	1	
15	1	0	

$$h_1 = \sigma(14 + 0) = 0.99$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

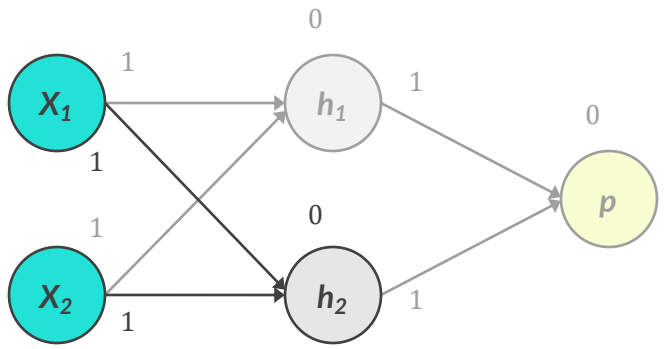
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)
14	**0**	**0**	
18	1	0	
22	0	0	
22	1	1	
26	0	1	
26	1	1	
30	0	1	
30	1	1	
35	0	1	
15	1	0	

$$h_1 = \sigma(14 + 0) = 0.99$$

$$h_2 = \sigma(14 + 0) = 0.99$$

$$p = \sigma(h_1 + h_2)$$

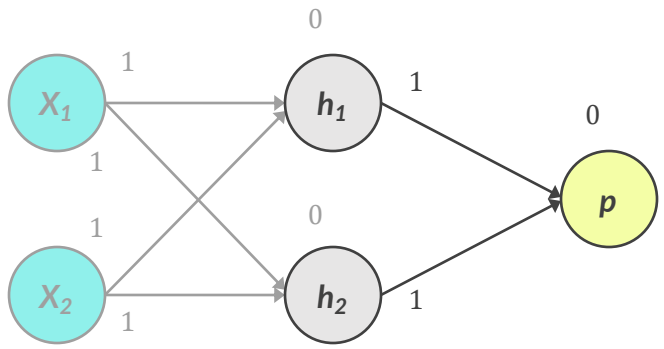
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)
14	**0**	**0**	
18	1	0	
22	0	0	
22	1	1	
26	0	1	
26	1	1	
30	0	1	
30	1	1	
35	0	1	
15	1	0	

$$h_1 = \sigma(14 + 0) = 0.99$$

$$h_2 = \sigma(14 + 0) = 0.99$$

$$p = \sigma(0.99 + 0.99) = 0.88$$

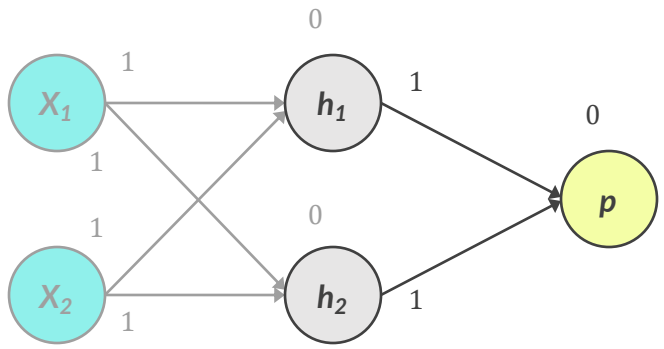
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)
14	**0**	**0**	**0.88**
18	1	0	
22	0	0	
22	1	1	
26	0	1	
26	1	1	
30	0	1	
30	1	1	
35	0	1	
15	1	0	

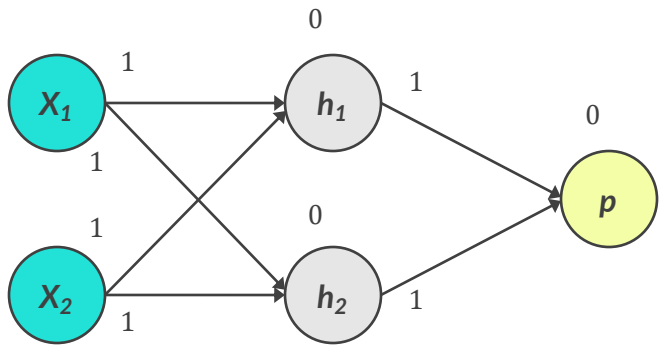
$$h_1 = \sigma(14 + 0) = 0.99$$

$$h_2 = \sigma(14 + 0) = 0.99$$

$$p = \sigma(0.99 + 0.99) = 0.88$$

TRAINING NEURAL NETWORKS

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)
14	**0**	**0**	**0.88**
18	1	0	
22	0	0	
22	1	1	
26	0	1	
26	1	1	
30	0	1	
30	1	1	
35	0	1	
15	1	0	

$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

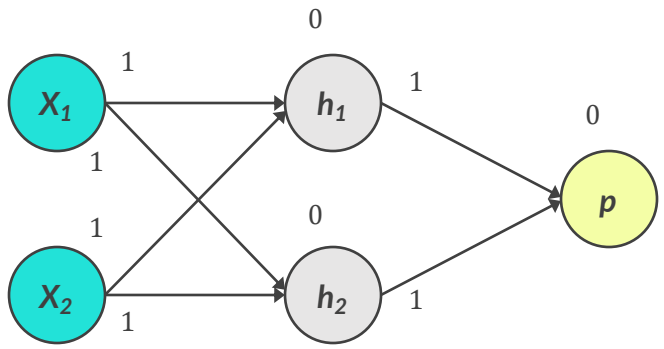
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 2: Forward pass –Starting from the left, apply all calculations through the neural network to get to a final set of predicted values



$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)
14	0	0	**0.88**
18	1	0	**0.88**
22	0	0	**0.88**
22	1	1	**0.88**
26	0	1	**0.88**
26	1	1	**0.88**
30	0	1	**0.88**
30	1	1	**0.88**
35	0	1	**0.88**
15	1	0	**0.88**

The initial model isn't sensitive to our inputs and predicts we'll be profitable 88% of the time

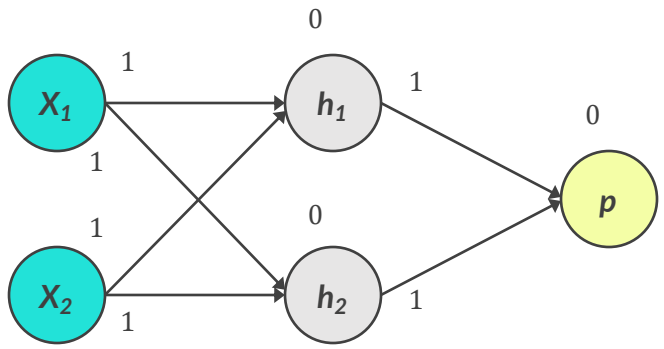
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 3: Calculate loss – Compare the predicted and actual values to compute the error, or loss



Training data:

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)
14	0	0	0.88
18	1	0	0.88
22	0	0	0.88
22	1	1	0.88
26	0	1	0.88
26	1	1	0.88
30	0	1	0.88
30	1	1	0.88
35	0	1	0.88
15	1	0	0.88

$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

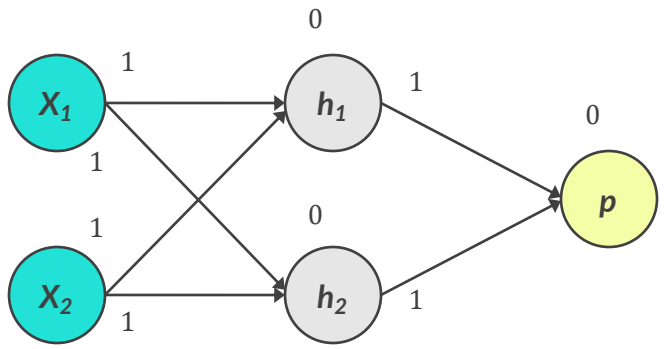
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 3: Calculate loss – Compare the predicted and actual values to compute the error, or loss



$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

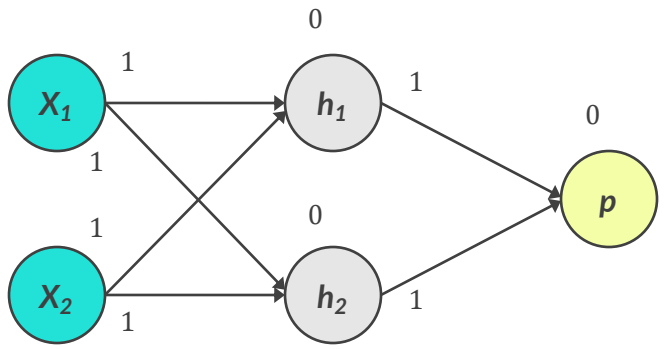
Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	**-0.88**
18	1	0	0.88	**-0.88**
22	0	0	0.88	**-0.88**
22	1	1	0.88	**0.12**
26	0	1	0.88	**0.12**
26	1	1	0.88	**0.12**
30	0	1	0.88	**0.12**
30	1	1	0.88	**0.12**
35	0	1	0.88	**0.12**
15	1	0	0.88	**-0.88**

LOG LOSS: **0.927**

TRAINING NEURAL NETWORKS

STEP 4: Update parameters – Starting from the right, calculate how much each parameter contributed to the loss with *back propagation*, and then use *gradient descent* to adjust the parameters by moving them a step closer to reducing the loss



$$h_1 = \sigma(x_1 + x_2)$$

$$h_2 = \sigma(x_1 + x_2)$$

$$p = \sigma(h_1 + h_2)$$

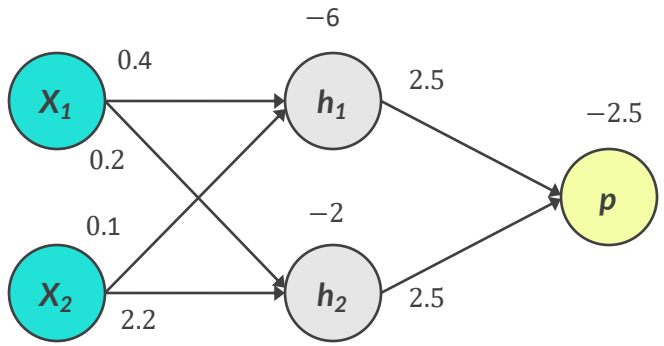
Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	-0.88
18	1	0	0.88	-0.88
22	0	0	0.88	-0.88
22	1	1	0.88	0.12
26	0	1	0.88	0.12
26	1	1	0.88	0.12
30	0	1	0.88	0.12
30	1	1	0.88	0.12
35	0	1	0.88	0.12
15	1	0	0.88	-0.88

LOG LOSS: **0.927**

TRAINING NEURAL NETWORKS

STEP 4: Update parameters – Starting from the right, calculate how much each parameter contributed to the loss with *back propagation*, and then use *gradient descent* to adjust the parameters by moving them a step closer to reducing the loss



Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	-0.88
18	1	0	0.88	-0.88
22	0	0	0.88	-0.88
22	1	1	0.88	0.12
26	0	1	0.88	0.12
26	1	1	0.88	0.12
30	0	1	0.88	0.12
30	1	1	0.88	0.12
35	0	1	0.88	0.12
15	1	0	0.88	-0.88

LOG LOSS: 0.927

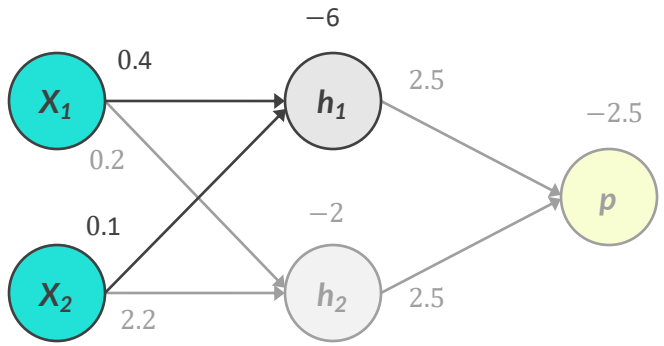
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 4: Update parameters – Starting from the right, calculate how much each parameter contributed to the loss with *back propagation*, and then use *gradient descent* to adjust the parameters by moving them a step closer to reducing the loss



$$h_1 = \sigma(0.4x_1 + 0.1x_2 - 6)$$

Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	-0.88
18	1	0	0.88	-0.88
22	0	0	0.88	-0.88
22	1	1	0.88	0.12
26	0	1	0.88	0.12
26	1	1	0.88	0.12
30	0	1	0.88	0.12
30	1	1	0.88	0.12
35	0	1	0.88	0.12
15	1	0	0.88	-0.88

LOG LOSS: **0.927**

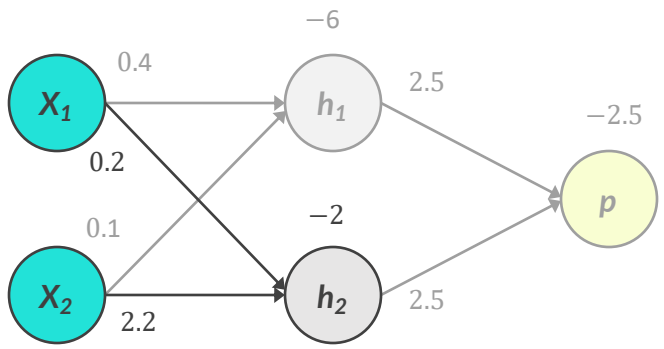
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 4: Update parameters – Starting from the right, calculate how much each parameter contributed to the loss with *back propagation*, and then use *gradient descent* to adjust the parameters by moving them a step closer to reducing the loss



$$h_1 = \sigma(0.4x_1 + 0.1x_2 - 6)$$

$$h_2 = \sigma(0.2x_1 + 2.2x_2 - 2)$$

Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	-0.88
18	1	0	0.88	-0.88
22	0	0	0.88	-0.88
22	1	1	0.88	0.12
26	0	1	0.88	0.12
26	1	1	0.88	0.12
30	0	1	0.88	0.12
30	1	1	0.88	0.12
35	0	1	0.88	0.12
15	1	0	0.88	-0.88

LOG LOSS: **0.927**

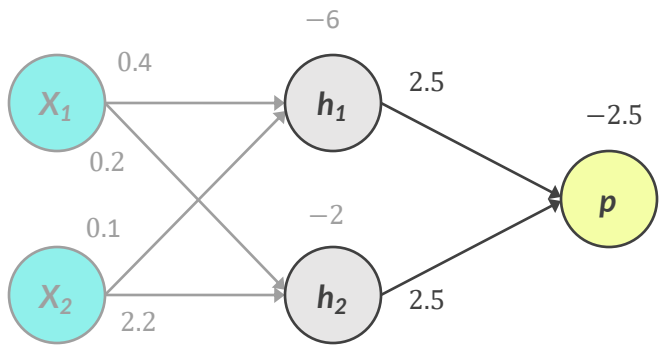
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 4: Update parameters – Starting from the right, calculate how much each parameter contributed to the loss with *back propagation*, and then use *gradient descent* to adjust the parameters by moving them a step closer to reducing the loss



$$h_1 = \sigma(0.4x_1 + 0.1x_2 - 6)$$

$$h_2 = \sigma(0.2x_1 + 2.2x_2 - 2)$$

$$p = \sigma(2.5h_1 + 2.5h_2 - 2.5)$$

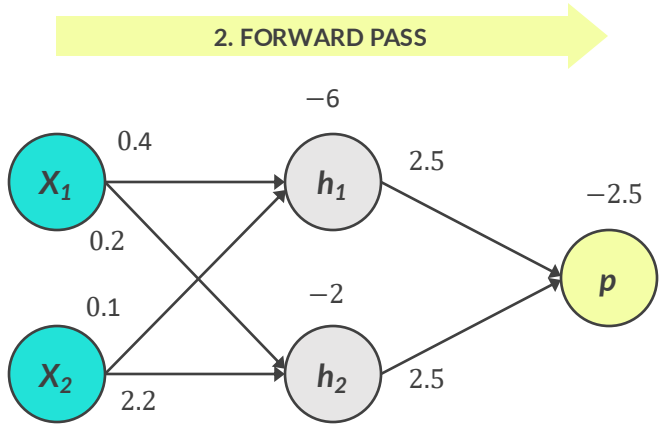
Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	-0.88
18	1	0	0.88	-0.88
22	0	0	0.88	-0.88
22	1	1	0.88	0.12
26	0	1	0.88	0.12
26	1	1	0.88	0.12
30	0	1	0.88	0.12
30	1	1	0.88	0.12
35	0	1	0.88	0.12
15	1	0	0.88	-0.88

LOG LOSS: **0.927**

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.4x_1 + 0.1x_2 - 6)$$

$$h_2 = \sigma(0.2x_1 + 2.2x_2 - 2)$$

$$p = \sigma(2.5h_1 + 2.5h_2 - 2.5)$$

Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.88	-0.88
18	1	0	0.88	-0.88
22	0	0	0.88	-0.88
22	1	1	0.88	0.12
26	0	1	0.88	0.12
26	1	1	0.88	0.12
30	0	1	0.88	0.12
30	1	1	0.88	0.12
35	0	1	0.88	0.12
15	1	0	0.88	-0.88

LOG LOSS: **0.927**

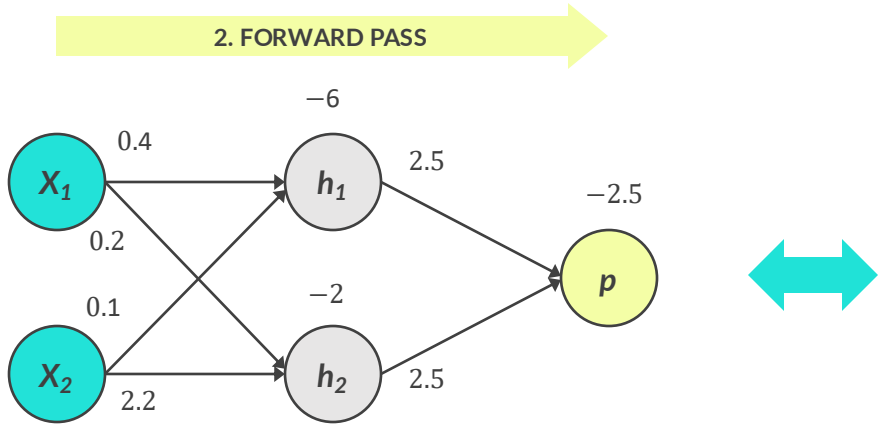
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.4x_1 + 0.1x_2 - 6)$$

$$h_2 = \sigma(0.2x_1 + 2.2x_2 - 2)$$

$$p = \sigma(2.5h_1 + 2.5h_2 - 2.5)$$

Training data:

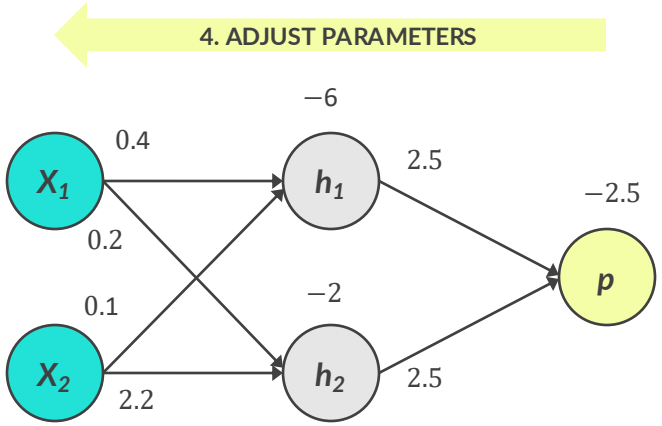
3. CALCULATE LOSS

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	**0.56**	
18	1	0	**0.87**	
22	0	0	**0.9**	
22	1	1	**0.91**	
26	0	1	**0.91**	
26	1	1	**0.92**	
30	0	1	**0.92**	
30	1	1	**0.92**	
35	0	1	**0.92**	
15	1	0	**0.77**	

This model still estimates we'll likely be profitable in each scenario, but the probabilities make more sense

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.4x_1 + 0.1x_2 - 6)$$

$$h_2 = \sigma(0.2x_1 + 2.2x_2 - 2)$$

$$p = \sigma(2.5h_1 + 2.5h_2 - 2.5)$$

Training data:

3. CALCULATE LOSS

Temperature (x_1)	*Weekend* (x_2)	*Profitable* (y)	*Prediction* (p)	*Error* (ε)
14	0	0	0.56	**-0.56**
18	1	0	0.87	**-0.87**
22	0	0	0.9	**-0.9**
22	1	1	0.91	**0.09**
26	0	1	0.91	**0.09**
26	1	1	0.92	**0.08**
30	0	1	0.92	**0.08**
30	1	1	0.92	**0.08**
35	0	1	0.92	**0.08**
15	1	0	0.77	**-0.77**

LOG LOSS: **0.710**

This is down from 0.927!

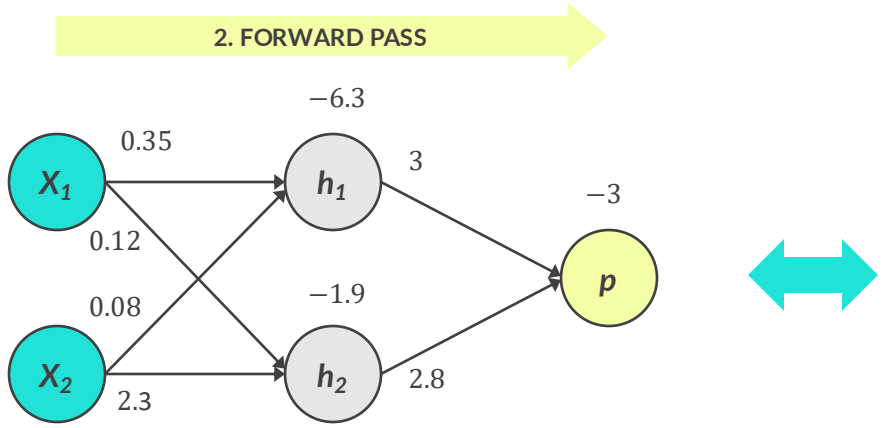
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.56	**-0.56**
18	1	0	0.87	**-0.87**
22	0	0	0.9	**-0.9**
22	1	1	0.91	**0.09**
26	0	1	0.91	**0.09**
26	1	1	0.92	**0.08**
30	0	1	0.92	**0.08**
30	1	1	0.92	**0.08**
35	0	1	0.92	**0.08**
15	1	0	0.77	**-0.77**

$$h_1 = \sigma(0.35x_1 + 0.08x_2 - 6.3)$$

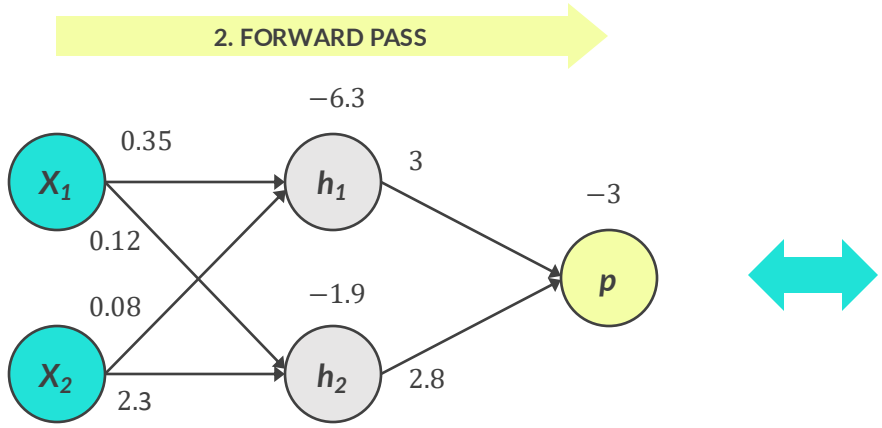
$$h_2 = \sigma(0.12x_1 + 2.3x_2 - 1.9)$$

$$p = \sigma(3h_1 + 2.8h_2 - 3)$$

LOG LOSS: **0.710**

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.35x_1 + 0.08x_2 - 6.3)$$

$$h_2 = \sigma(0.12x_1 + 2.3x_2 - 1.9)$$

$$p = \sigma(3h_1 + 2.8h_2 - 3)$$

Training data:

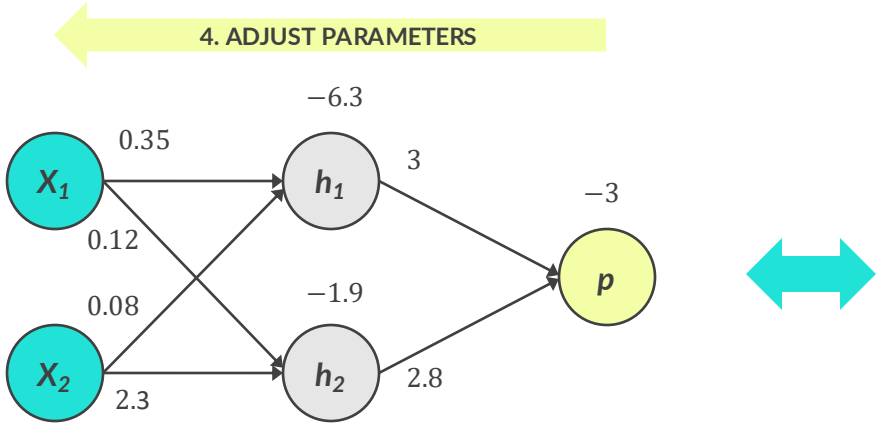
3. CALCULATE LOSS

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	**0.24**	
18	1	0	**0.76**	
22	0	0	**0.79**	
22	1	1	**0.89**	
26	0	1	**0.88**	
26	1	1	**0.93**	
30	0	1	**0.91**	
30	1	1	**0.94**	
35	0	1	**0.93**	
15	1	0	**0.59**	

We're now predicting we likely won't be profitable in low temperature weekdays, which makes sense!

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.35x_1 + 0.08x_2 - 6.3)$$

$$h_2 = \sigma(0.12x_1 + 2.3x_2 - 1.9)$$

$$p = \sigma(3h_1 + 2.8h_2 - 3)$$

Training data:

3. CALCULATE LOSS

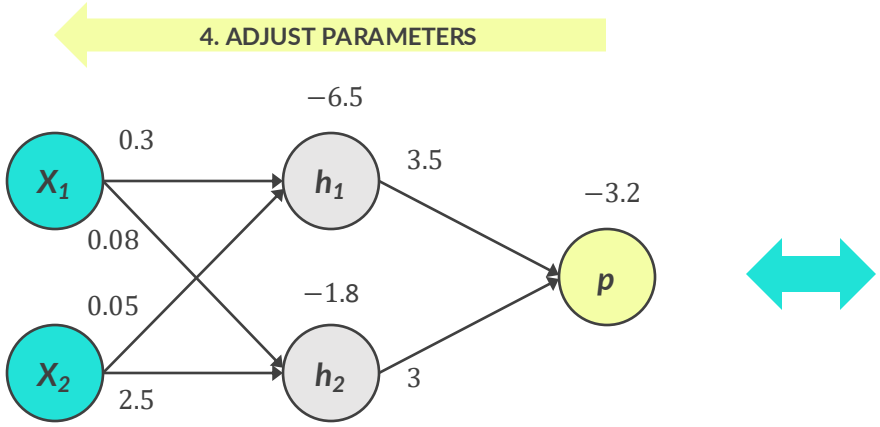
Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.24	**-0.24**
18	1	0	0.76	**-0.76**
22	0	0	0.79	**-0.79**
22	1	1	0.89	**0.11**
26	0	1	0.88	**0.12**
26	1	1	0.93	**0.07**
30	0	1	0.91	**0.09**
30	1	1	0.94	**0.06**
35	0	1	0.93	**0.07**
15	1	0	0.59	**-0.59**

LOG LOSS: **0.468**

This is down from 0.710!

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.24	-0.24
18	1	0	0.76	-0.76
22	0	0	0.79	-0.79
22	1	1	0.89	0.11
26	0	1	0.88	0.12
26	1	1	0.93	0.07
30	0	1	0.91	0.09
30	1	1	0.94	0.06
35	0	1	0.93	0.07
15	1	0	0.59	-0.59

$$h_1 = \sigma(0.3x_1 + 0.05x_2 - 6.5)$$

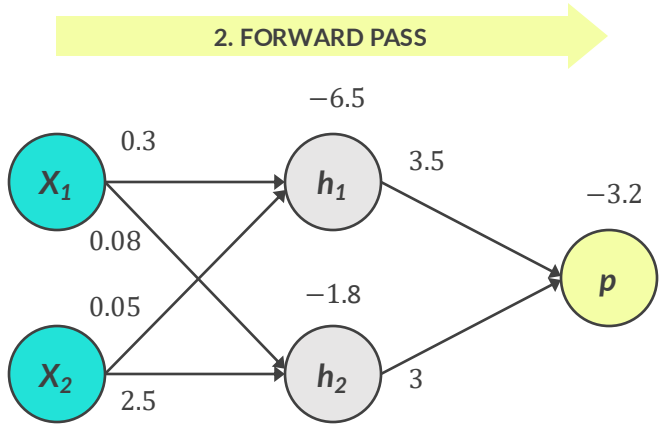
$$h_2 = \sigma(0.08x_1 + 2.5x_2 - 1.8)$$

$$p = \sigma(3.5h_1 + 3h_2 - 3.2)$$

LOG LOSS: **0.468**

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.24	-0.24
18	1	0	0.76	-0.76
22	0	0	0.79	-0.79
22	1	1	0.89	0.11
26	0	1	0.88	0.12
26	1	1	0.93	0.07
30	0	1	0.91	0.09
30	1	1	0.94	0.06
35	0	1	0.93	0.07
15	1	0	0.59	-0.59

$$h_1 = \sigma(0.3x_1 + 0.05x_2 - 6.5)$$

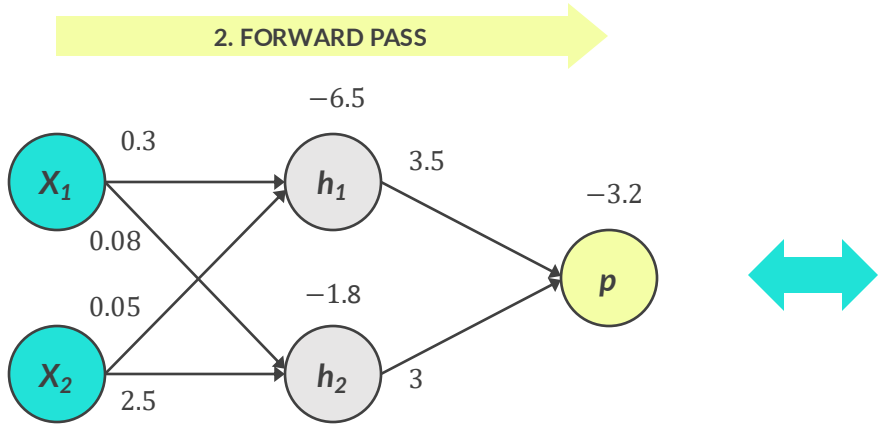
$$h_2 = \sigma(0.08x_1 + 2.5x_2 - 1.8)$$

$$p = \sigma(3.5h_1 + 3h_2 - 3.2)$$

LOG LOSS: **0.468**

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.3x_1 + 0.05x_2 - 6.5)$$

$$h_2 = \sigma(0.08x_1 + 2.5x_2 - 1.8)$$

$$p = \sigma(3.5h_1 + 3h_2 - 3.2)$$

Training data:

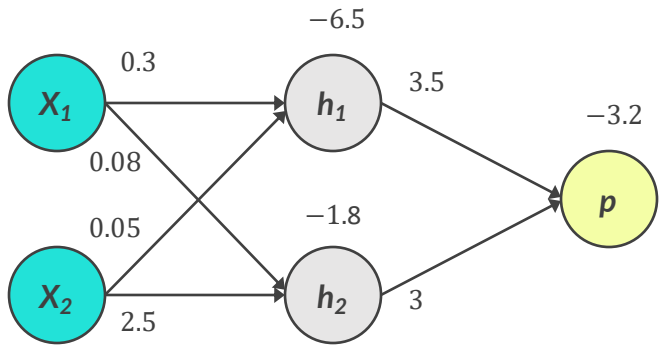
3. CALCULATE LOSS

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	**0.13**	
18	1	0	**0.6**	
22	0	0	**0.53**	
22	1	1	**0.81**	
26	0	1	**0.78**	
26	1	1	**0.92**	
30	0	1	**0.88**	
30	1	1	**0.95**	
35	0	1	**0.92**	
15	1	0	**0.46**	

The probabilities for profit are much more spread out and sensitive to both input features!

TRAINING NEURAL NETWORKS

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.3x_1 + 0.05x_2 - 6.5)$$

$$h_2 = \sigma(0.08x_1 + 2.5x_2 - 1.8)$$

$$p = \sigma(3.5h_1 + 3h_2 - 3.2)$$

Training data:

3. CALCULATE LOSS

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.13	**-0.13**
18	1	0	0.6	**-0.6**
22	0	0	0.53	**-0.53**
22	1	1	0.81	**0.19**
26	0	1	0.78	**0.22**
26	1	1	0.92	**0.08**
30	0	1	0.88	**0.12**
30	1	1	0.95	**0.05**
35	0	1	0.92	**0.08**
15	1	0	0.46	**-0.46**

LOG LOSS: **0.323**

This is now optimized!

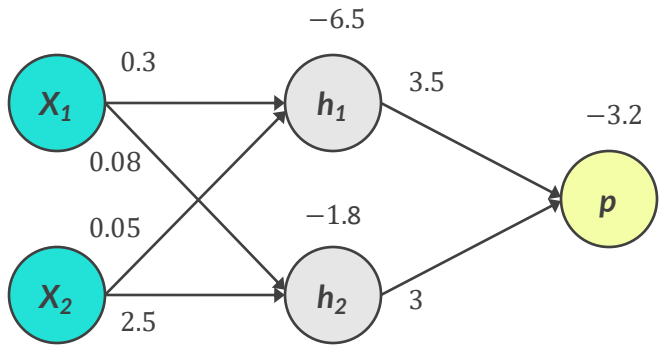
TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

STEP 5: Repeat – Repeat steps 2-4 until you minimize the loss or reach an iteration limit and lock in the final model parameters (weights and biases)



$$h_1 = \sigma(0.3x_1 + 0.05x_2 - 6.5)$$

$$h_2 = \sigma(0.08x_1 + 2.5x_2 - 1.8)$$

$$p = \sigma(3.5h_1 + 3h_2 - 3.2)$$

Training data:

Temperature (x_1)	Weekend (x_2)	Profitable (y)	Prediction (p)	Error (ε)
14	0	0	0.13	-0.13
18	1	0	0.6	-0.6
22	0	0	0.53	-0.53
22	1	1	0.81	0.19
26	0	1	0.78	0.22
26	1	1	0.92	0.08
30	0	1	0.88	0.12
30	1	1	0.95	0.05
35	0	1	0.92	0.08
15	1	0	0.46	-0.46

Today *(new data)*:

25	0	**0.93**	***Profitable!***

EXERCISE: TRAINING NEURAL NETWORKS

Modern NLP Overview

Neural Networks

Deep Learning

Place these steps in the correct order:

- Adjust parameters
- Forward pass
- Random parameters
- Loss calculation
- Optimal parameters

1.
2.
3.
4.
5.

Answer these questions:

What do you need to specify before training?

ANSWER BELOW

What do you get at the end of training?

ANSWER BELOW

How do you train a NN model in practice?

ANSWER BELOW

DEEP LEARNING

Modern NLP Overview

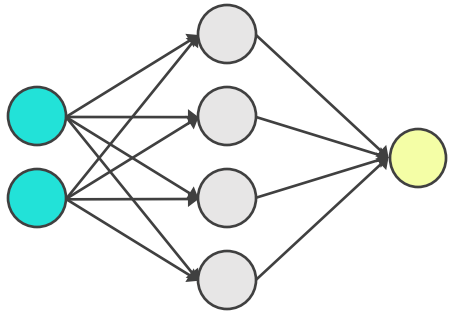
Neural Networks

Deep Learning

Deep learning refers to a neural network with 3 or more hidden layers

- Deep learning has revolutionized the field of artificial intelligence (including natural language processing, computer vision, speech recognition, and more) since the 2010s
- While the math has been around since the 1950s (invention of logistic regression and perceptron), the computational power of the 2010s has boosted to them to new heights

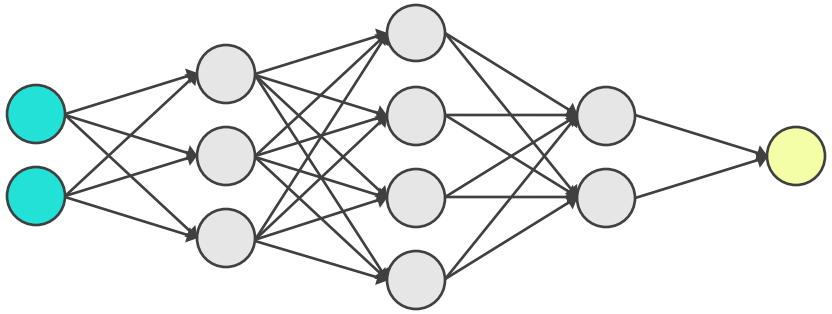
Neural Network



Sometimes used for:

- Simple classification tasks *(0 or 1)*
- Medium data sets *(thousands of rows)*

Deep Learning



Often used for:

- More complex tasks *(NLP, CV, ASR, etc.)*
- Large data sets *(millions of rows)*

DEEP LEARNING ARCHITECTURES

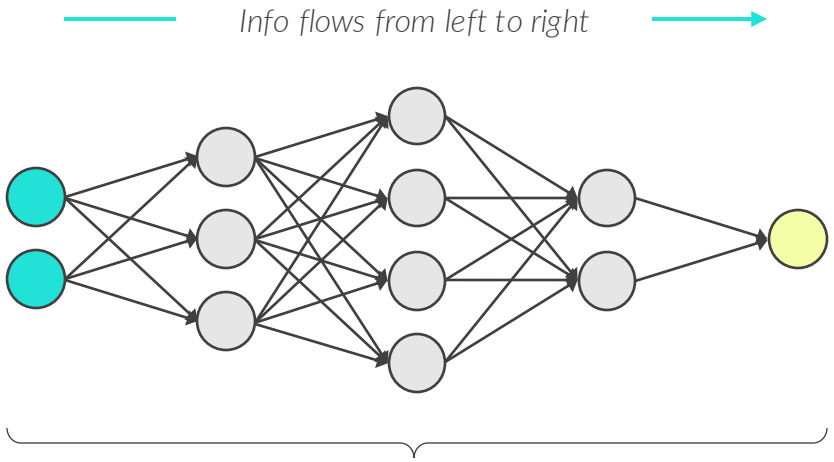
Modern NLP Overview

Neural Networks

Deep Learning

Deep learning architectures combine deep learning with additional calculations and specialized operations to perform the ground-breaking tasks

- What we've been working with so far is a **Feedforward Neural Network (FNN)**, which is the simplest deep learning architecture *(no additional calculations or operations)*



When every node is connected to every other node, it's called a ***fully connected neural network***

FNNs are often a piece of other, more complex, deep learning architectures

(this is an important piece of the next section on Transformers!)

DEEP LEARNING ARCHITECTURES

Modern NLP Overview

Neural Networks

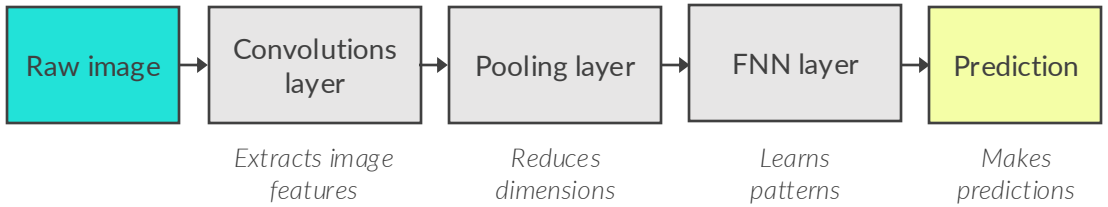
Deep Learning

These are some of the most popular **deep learning architectures** and their layers:

Convolutional Neural Networks (CNNs)

Popularized in 2012

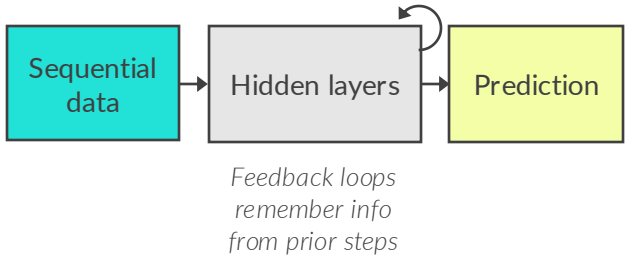
Applications:
Image-related tasks like image classification, object detection, etc.



Recurrent Neural Networks (RNNs)

Popularized in 2013

Applications:
Sequential-tasks, such as NLP tasks, time series analysis, etc.

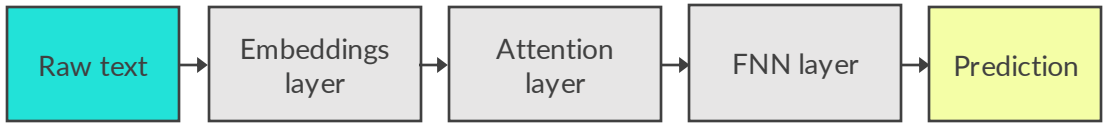


Long Short-Term Memory (LSTMs) *are an extension of RNNs that include logic to remember and forget info over time (popularized in 2015)*

Transformers

Popularized in 2017

Applications:
NLP, CV, ASR tasks, and much more!



(More on these in the Transformers section up next!)

Transformers have replaced RNNs and LSTMs for NLP tasks

DEEP LEARNING ARCHITECTURES

Modern NLP Overview

Neural Networks

Deep Learning

Summary of **deep learning architectures**:

Architecture	Description	Application
Feedforward Neural Network (FNN)	Basic fully connected neural network	Building block for other deep learning architectures
Convolutional Neural Network (CNN)	Extract image features using convolutions, then feed into FNN	Used for image tasks
Recurrent Neural Network (RNN) / Long Short-Term Memory (LSTM)	Use output as input for next prediction	Used for sequential tasks *(but not NLP tasks anymore)*
Transformers	Consists of embeddings, attention, and FNN layers (*coming up next!*)	Used for NLP, computer vision, speech recognition, etc.

DEEP LEARNING IN PRACTICE

Modern NLP Overview

Neural Networks

Deep Learning

Traditional NLP way of thinking: train your model

1. Pick a **model** that's good for your problem *(predict if a company will be profitable)*
2. Provide your historical data *(inputs = company descriptions, labels = profitable 1/0)*
3. Feed it into model to get final parameters – now the model is trained
4. Make predictions using this **trained model**

DOWNSIDES: To fit a deep learning model, you need millions of rows of labeled data, along with significant computational resources

Modern NLP way of thinking: use a pretrained model *(parameters are locked-in)*

1. Pick a **pretrained model** that's good for your problem *(predict if company will be profitable)*
2. Make predictions using this **pretrained model**
3. (Optional) Improve the predictions using transfer learning or fine-tuning

NOTE: Only research labs and large tech companies will train their own deep learning models from scratch these days, while the majority of data scientists use or start with pretrained models

DEEP LEARNING IN PRACTICE

Modern NLP Overview

Neural Networks

Deep Learning

Most data scientists will use **pretrained deep learning models** for their analysis

- These pretrained models have already been trained on extremely large data sets, so all the parameters (weights, biases, etc.) are locked in
- **Large Language Models (LLMs)** are deep learning models that are pretrained on massive amounts of text data, including BERT and GPT *(much more on this in the next section!)*
- To use an LLM, you input your text, and then all the calculations (weighted sums, non-linear transformations, etc.) are applied to output a final prediction

This is a **big mindset shift** for data scientists.

Traditionally, data scientists have been focused on model interpretability, always knowing what's happening behind the scenes and avoiding using black box techniques and falling into the "danger zone".

With NLP tasks these days, pretrained deep learning models work so well that they're the gold standard for NLP tasks, and even though it's a black box, it's absolutely acceptable and encouraged to use them.

The goal of this section and the next is to break down the components of popular deep learning models at a high level to give you an idea of how they work before applying them using Hugging Face.

PRETRAINED DEEP LEARNING MODELS

Modern NLP Overview

Neural Networks

Deep Learning

There are multiple ways to use a **pretrained deep learning model**:

These two are covered in the Hugging Face Transformers section of this course

Pretrained model only

Download and use a pretrained model as is to make predictions

- Parameters are fixed
- Used for sentiment analysis, text summarization, etc.

Pretrained model embeddings

Use a pretrained model's embeddings as inputs into traditional machine learning models

- Parameters are fixed
- Used for document similarity, document clustering, etc.

Pretrained model with transfer learning

Start with a pretrained model and adjust the parameters by training on task / domain-specific data*

- Parameters are updated in final layers or all layers
- Used for text classification, industry-specific analysis, etc.

**Adjusting weights requires a large amount of data (at least tens of thousands of labeled data points) & computational power (more than a single computer, many GPUs)*

Pretrained model with RAGs

Combine pretrained models with external databases to be more up-to-date and context-aware**

- Parameters may or may not be updated
- Used for question answering, fact checking, etc.

***RAGs (Retrieval Augmented Generation) require building a structured retrieval database to hold at least tens of thousands of external text documents*

EXERCISE: DEEP LEARNING

Modern NLP Overview

Neural Networks

Deep Learning

Match each term with its definition:

Traditional NLP	Modern NLP	Transformers
CNNs	RNNs / LSTMs	FNNs
Neural networks	Deep learning	

	1-2 hidden layers
	3+ hidden layers
	Building block for other architectures
	Used for image-related tasks
	Used for NLP tasks, but is outdated
	Used for NLP tasks
	Train your own model
	Use a pretrained model

Order these from easiest to hardest:

- Training a model from scratch
- Fine-tuning a pretrained model
- Using a pretrained model as is
- Using embeddings from a pretrained model

Easiest
Hardest

KEY TAKEAWAYS

Neural networks are ML models with input, hidden, and output layers

- *They are sometimes called artificial neural networks (ANNs) or multilayer perceptrons (MLPs)*
- *At each node, the weighted sum of the inputs is calculated and a non-linear transformation applied*
- *To train a neural network, start with random parameters and slowly adjust them until they become optimal*

Deep learning refers to a neural network with three or more hidden layers

- *DL is often used for more complex applications such as NLP, computer vision, speech recognition, etc.*

Deep learning architectures combine deep learning with extra calculations

- *Popular architectures include Transformers for natural language processing, CNNs for computer vision, etc.*
- *These architectures often include basic FNNs, with additional modifications based on the input data (feature extraction for image data, loops for sequential data, embeddings and attention for text data, etc.)*

Most data professionals use **pretrained deep learning models** for analysis

- *Pretrained models (set parameters) are trained on millions of data points and perform well out-of-the-box*
- *AI / ML researchers will train and sometimes data scientists will fine-tune models for domain-specific data sets*

TRANSFORMERS & LLMS

TRANSFORMERS & LLMS

In this section, we'll introduce **transformers** and its main layers, as well as pretrained deep learning models specifically for NLP tasks: **large language models (LLMs)**

TOPICS WE'LL COVER:

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

GOALS FOR THIS SECTION:

- Become familiar with the main components of the transformer architecture: embeddings, attention, and feedforward neural networks
- Review the differences between encoder-only, decoder-only and encoder-decoder models
- Get introduced to popular large language models, including BERT, GPT, and more

RECAP: MODERN NLP CONCEPTS

In this section, we'll be covering the rest of these **modern NLP concepts** to understand how LLMs work before applying them in Hugging Face:

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

COMPLEXITY

Concepts	Key Terms
1 Neural Networks & Deep Learning a) Logistic Regression b) Neural Networks ← *We've covered these now!* → c) Deep Learning	• *Neural network components: layers, nodes, weights, parameters, activation functions* • *Neural network training: forward pass, loss, backpropagation, gradient descent* • *Deep learning architectures: FNN, CNN, RNN, LSTM, Transformers*
2 Transformers & LLMs a) Embeddings b) Attention c) Transformer-Based LLMs	• *Embeddings: tokens* • *Attention: queries, keys, scores* • *Feedforward neural network* • *Transformers: encoders vs decoders* • *Pretrained LLMs: BERT, GPT and more*

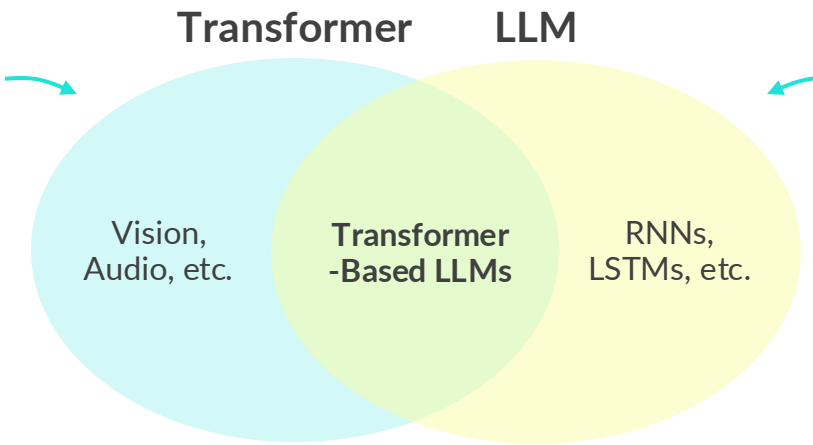
TRANSFORMERS & LLMS

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

Transformers are a deep learning architecture with three main layers: embeddings, attention, and feedforward neural networks (FNN)

Large Language Models (LLMs) are deep learning models that have been pretrained on a massive amount of text data

*Transformers can be used for many tasks, but are **mainly used for NLP applications***



*LLMs can be based on many deep learning architectures, but they are **mainly based on transformers***

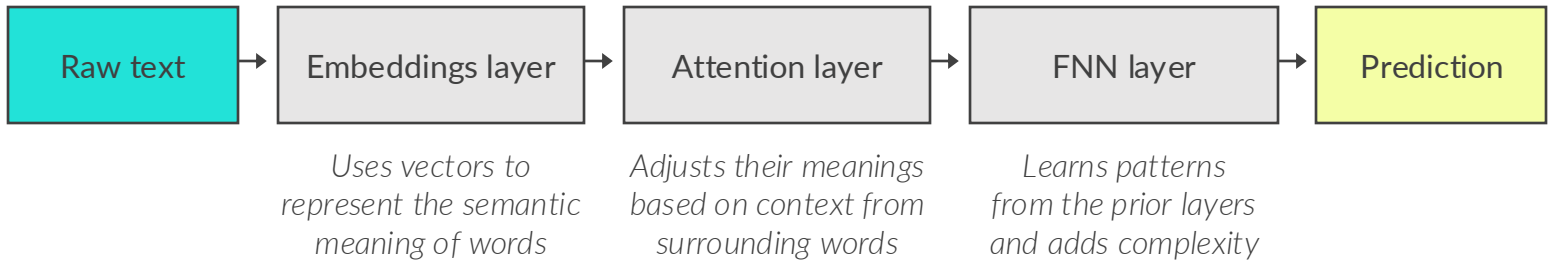
***Transformer-Based LLMs** are the most popular DL approach to NLP tasks*

TRANSFORMER ARCHITECTURE

The **transformer architecture** refers to the series of layers and computations that the input data passes through to produce a final result

- Along the way, the input text is gradually *transformed*, hence the name transformers

These are the **main layers** of a transformer:



EMBEDDINGS

The first layer of a transformer, the **embeddings** layer, converts text tokens into meaningful numeric representations

- It places each token (word) into a high-dimensional space, so words with similar meanings end up close together, and words with different meanings are farther apart

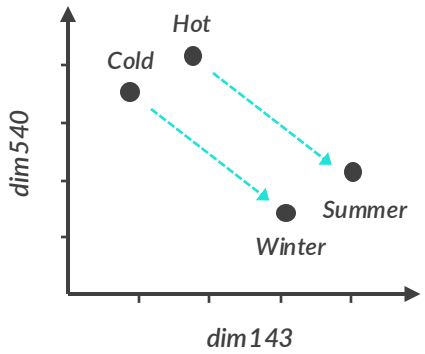
"I love cold lemonade!"

Embeddings layer:

token	dim1	dim2	...	dim768
I	0.16	-0.04	...	0.67
love	-0.21	0.59	...	0.33
cold	0.04	-0.14	...	0.89
lemonade	-0.11	0.35	...	-0.15
!	0.05	-0.03	...	-0.06

Total dimensions vary, but 768 is a common length for LLMs

Tokens include things like punctuation!



*The vector (768 numbers) for each token represents its location in space – **amazingly, these values have semantic meaning**!*

EMBEDDINGS

Transformers & LLMs

Embeddings

Attention

FNNs

Encoders & Decoders

Pretrained LLMs

The first layer of a transformer, the **embeddings** layer, converts text tokens into meaningful numeric representations

- It places each token (word) into a high-dimensional space, so words with similar meanings end up close together, and words with different meanings are farther apart

How are these values generated?

- Popular word embeddings are trained using shallow neural networks (word2vec) and matrix factorization (GloVe) (*both are concepts you're now familiar with!)*
- Within an LLM though, these values (weights) are randomly initialized and slowly updated until they reach their final values *(like we saw in the Neural Networks section)*
- Remember when we talked about how all the parameters of a neural network are represented as matrices in Python? This embeddings matrix here is exactly that!

In the embedding layer alone, given a vocabulary size of 50k and 768 dimensions for each token, the embedding matrix would have 38 million parameters! And that's just the start of a transformer...

ATTENTION

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

The second layer of a transformer, the **attention** layer, adds context by helping each token absorb additional meaning from other tokens

Without the attention layer, the word "lemonade" is in the same location in space for all three of these sentences, even though it has a different meaning in each one

"lemonade"

This will be in an exact location based on the word embedding

"I love cold lemonade!"

This will be in a slightly different location, since it's specifically cold lemonade that's loved

"I love Beyonce's Lemonade album."

This will be in a very different location, since it's about an album, not a drink

ATTENTION

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

The second layer of a transformer, the **attention** layer, adds context by helping each token absorb additional meaning from other tokens

The token here is "lemonade"

"I love cold lemonade!" → *Without the attention layer, the meaning of the word "lemonade" isn't affected by the other words in the sentence*

*"I love **cold** lemonade!"* → *With the attention layer, the word "cold" **adds context to** "lemonade"*

- *In technical terms: cold **attends to** lemonade*
- *In layman's terms: this isn't just any lemonade, it's a cold one*

*"I **love** cold lemonade!"* → *With the attention layer, the word "love" **adds context to** "lemonade"*

- *In technical terms, love **attends to** lemonade*
- *In layman's terms: this isn't just any lemonade, it's lemonade that's loved*

"Lemonade" will absorb the meanings of "cold" and "love", but not so much "I"

ATTENTION

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

The second layer of a transformer, the **attention** layer, adds context by helping each token absorb additional meaning from other tokens

- It does this by creating matrices for **queries**, **keys**, and **attention scores**

Queries: *Questions about other tokens*

token	q1	q2	...	q768
I	0.12	-0.30	...	0.08
love	0.85	0.14	...	-0.22
cold	0.23	-0.10	...	0.16
lemonade	0.11	0.22	...	0.03
!	-0.05	0.09	...	-0.01

This asks: "Who or what do I love?"

Keys: *Answers to those questions*

token	k1	k2	...	k768
I	-0.02	-0.25	...	0.04
love	0.18	0.10	...	-0.20
cold	0.42	-0.05	...	0.10
lemonade	0.89	0.19	...	0.01
!	-0.10	0.05	...	-0.02

This says: "I can be loved!"

The queries and keys here are one of many query-key pairs in a transformer. Other queries about love could be "what is expressing the love?", "what kind of love is being expressed?", etc.

ATTENTION

The second layer of a transformer, the **attention** layer, adds context by helping each token absorb additional meaning from other tokens

- It does this by creating matrices for **queries**, **keys**, and **attention scores**

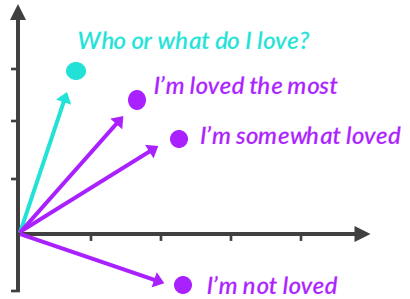
Queries: *Questions about other tokens*

token	q1	q2	...	q768
I	0.12	-0.30	...	0.08
love	0.85	0.14	...	-0.22
cold	0.23	-0.10	...	0.16
lemonade	0.11	0.22	...	0.03
!	-0.05	0.09	...	-0.01

This asks: "Who or what do I love?"

Keys: *Answers to those questions*

token	k1	k2	...	k768	
I	-0.02	-0.25	...	0.04	
love	0.18	0.10	...	-0.20	
cold	0.42	-0.05	...	0.10	*I'm somewhat loved*
lemonade	0.89	0.19	...	0.01	*I'm loved the most*
!	-0.10	0.05	...	-0.02	



*All these relationships are summarized in an **attention scores** matrix (up next!)*

ATTENTION

The second layer of a transformer, the **attention** layer, adds context by helping each token absorb additional meaning from other tokens

- It does this by creating matrices for **queries**, **keys**, and **attention scores**

Attention scores: *Summary of query-key relationships*

	I	love	cold	lemonade	!
I	0.2	0.6	0	0	0.2
love	0.1	0.1	0.3	0.4	0.1
cold	0.1	0.1	0.2	0.5	0.1
lemonade	0.3	0.1	0.4	0.1	0.1
!	0.3	0.1	0.1	0	0.5

The "love" is mostly for "lemonade", and somewhat for "cold"

"Cold" mostly describes "lemonade"

"Lemonade" is getting "love" and is "cold"

ATTENTION

Transformers & LLMs

Embeddings

Attention

FNNs

Encoders & Decoders

Pretrained LLMs

The second layer of a transformer, the **attention** layer, adds context by helping each token absorb additional meaning from other tokens

- It does this by creating matrices for **queries**, **keys**, and **attention scores**

How are these values generated?

- Like embeddings, the query and key values are randomly initialized and slowly updated until they reach their final values
- What are the additional calculations?
  - To capture query-key similarity, a dot product (similarity score) is taken
  - For attention scores to add up to 1, a softmax normalization function is applied

Like how the embeddings layer amazingly captured word meaning, this attention layer amazingly captures how much each token attends to, or gives context to, other tokens

FEEDFORWARD NEURAL NETWORK

Transformers & LLMs

Embeddings

Attention

FNNs

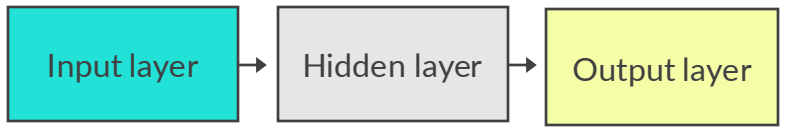
Encoders & Decoders

Pretrained LLMs

The third layer of a transformer, the **feedforward neural network (FNN)** layer, learns patterns from the data and adds complexity to the model

- Embedding layer: words are placed in locations in space that hold some meaning
- Attention layer: the locations are adjusted based on context from surrounding words
- FNN layer: patterns in those contextual relationships are learned and captured

Typical FNN in a transformer:



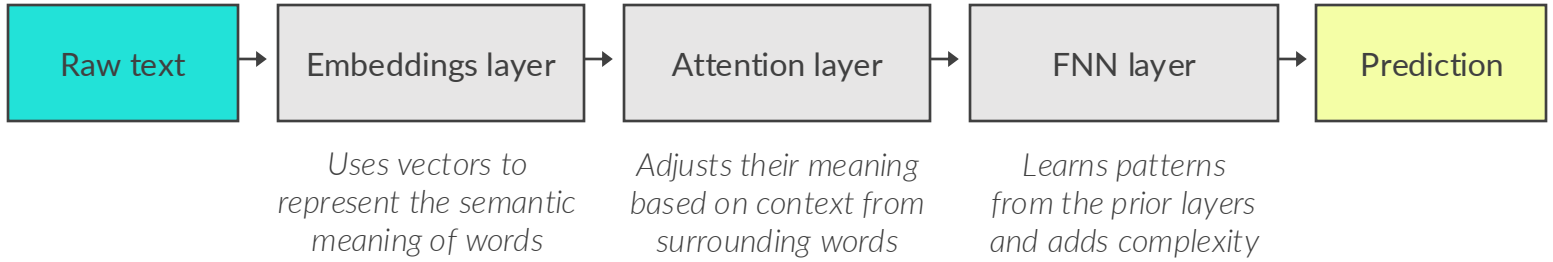
The output is a transformed representation of the original tokens with refined meanings

One of these nodes could be capturing the idea of a noun-adjective relationship, another could be capturing the idea of love, etc.

This is just a **widely-accepted theory** or interpretation of how this works to make it more understandable – the actual workings are more abstract and less interpretable

SUMMARY: TRANSFORMERS

To summarize these are the **main layers** of a transformer:

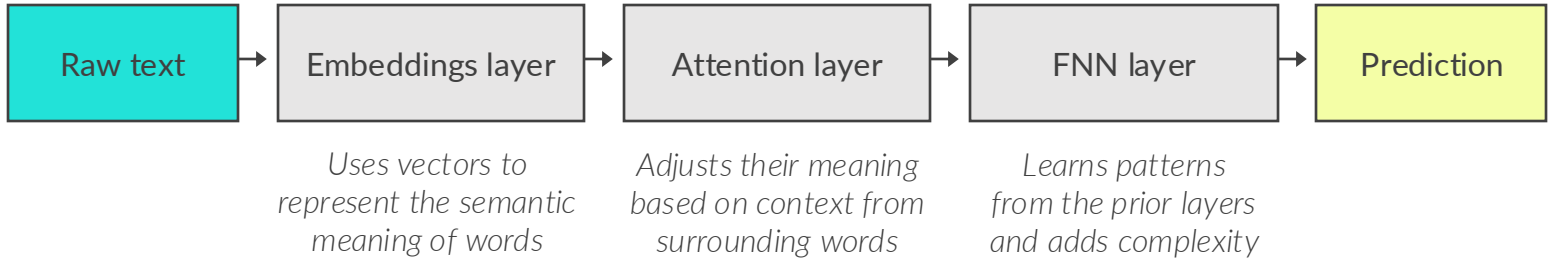


How are do transformers work so well? **Attention.**

- Attention captures **context** – it enriches the meaning of each word based on others
- Attention allows for **parallelization** – unlike RNNs & LSTMs which process words one at a time in a sequence, the attention step processes an entire sequence at once, making it highly parallelizable for training and able to handle huge data sets
- Attention **generalizes** well – this core transformers architecture works well for a variety of NLP tasks with little fine-tuning

SUMMARY: TRANSFORMERS

To summarize these are the **main layers** of a transformer:



How are do transformers work so well? **Attention.**

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

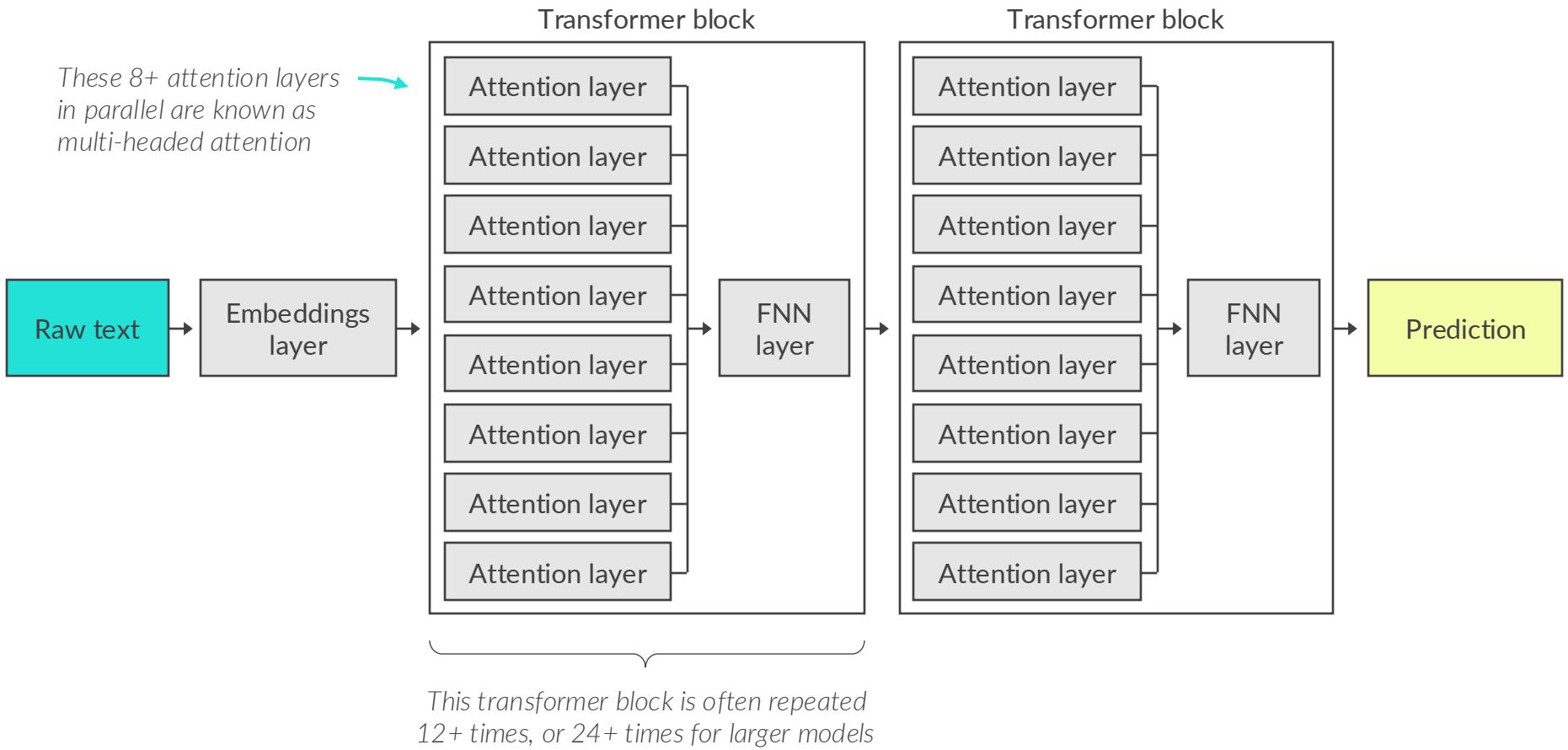
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

The famous 2017 paper was right!

SUMMARY: TRANSFORMERS

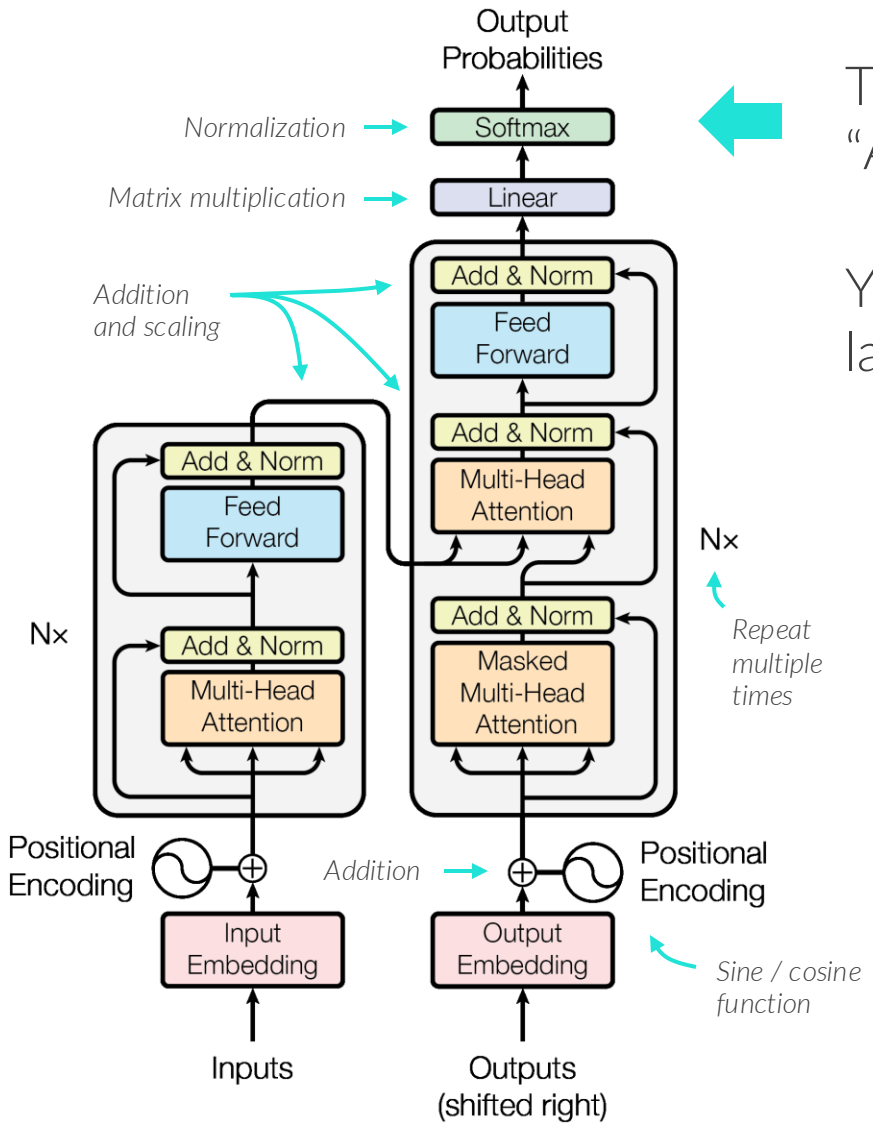
- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

In reality, the layers typically follow this order:



BREAKING DOWN THE TRANSFORMER DIAGRAM

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs



This is the **transformer diagram** from the "Attention is All You Need" paper (2017)

You now understand its three core learning layers that capture patterns in text data!

- **Embedding**
- **Attention**
- **Feedforward neural network**

The other components in the diagram are **math calculations** that support the learning process, but don't learn patterns themselves

Attention is All You Need paper https://arxiv.org/pdf/1706.03762

ENCODERS & DECODERS

There are three main **categories of transformers**: encoder-only models, decoder-only models, and encoder-decoder models

- Different models will use different pieces of the transformer architecture

Encoder-Only Models

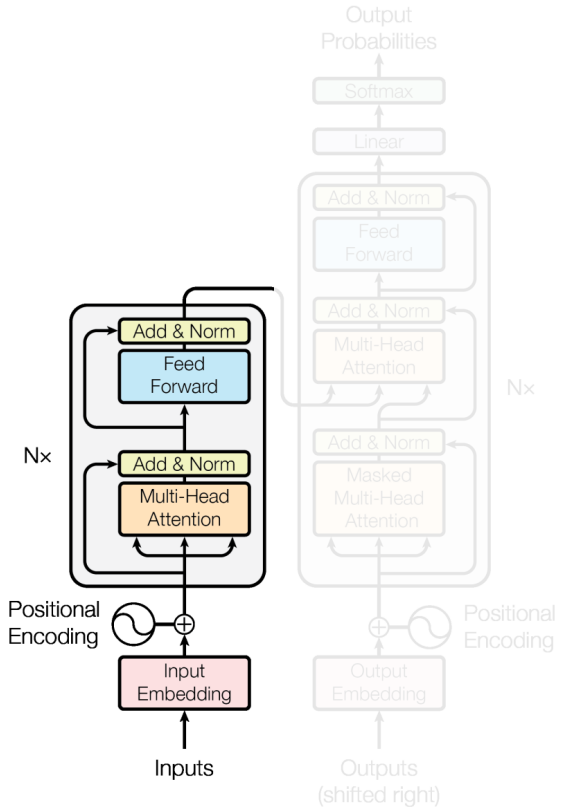
Only use the left side of the architecture, aka the encoder

The encoder takes raw text and encodes it as an embedding representation of the text

In short, ***it understands text***

Application: Sentiment Analysis

- **Input:** "I love cold lemonade!"
- **Output:** Positive



While encoders embed text, they can be fine-tuned for specific tasks like **sentiment analysis**, where an extra classification step is added to get from embedding to output

ENCODERS & DECODERS

There are three main **categories of transformers**: encoder-only models, decoder-only models, and encoder-decoder models

- Different models will use different pieces of this transformer architecture

Decoder-Only Models

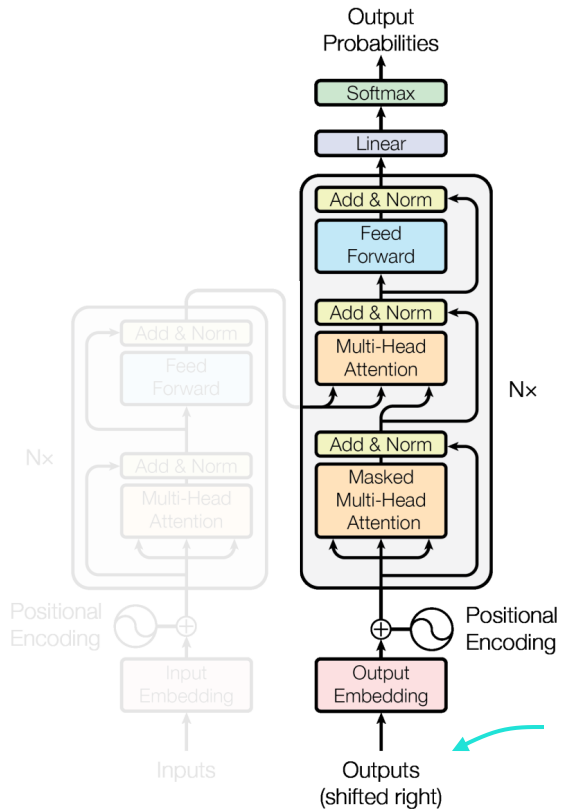
Only use the right side of the architecture, aka the decoder

The decoder takes an input text sequence and infers* the next word

In short, ***it generates text***

Application: Text Generation

- **Input:** "I love cold lemonade!"
- **Output:** "It's the perfect drink! 🍋🥤"



*For decoder-only models, these are **inputs***

*With transformers & LLMs, the word inference is typically used instead of the word prediction

ENCODERS & DECODERS

There are three main **categories of transformers**: encoder-only models, decoder-only models, and encoder-decoder models

- Different models will use different pieces of this transformer architecture

Encoder-Decoder Models

Use the entire architecture, both the encoder and decoder sides

The encoder-decoder takes two inputs:

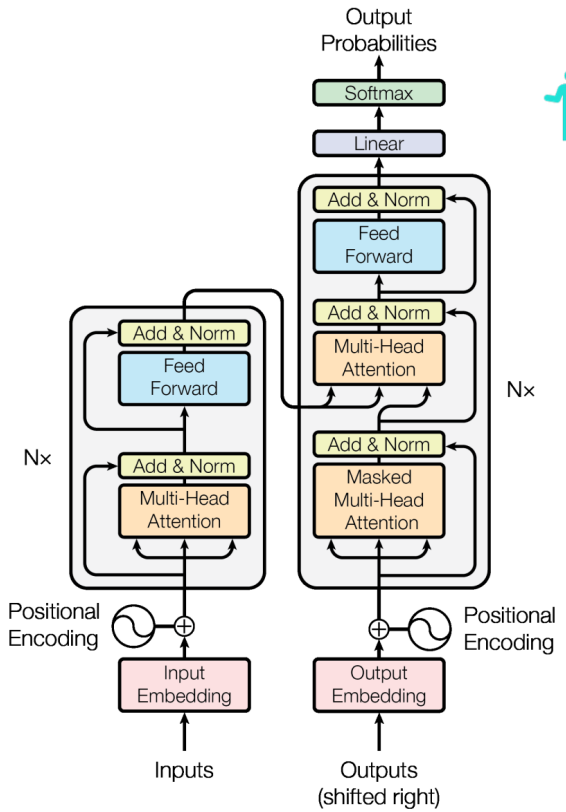
1. A text sequence
2. A shifted target sequence

Both are encoded as embeddings and combined to infer the next word

In short, ***it understands and generates text***

Application: Translation

- **Input:** "I love cold lemonade!"
- **Output:** "¡Me encanta la limonada fría!"



How do we use these in practice?

- Download a pretrained LLM *(coming up next!)*

LARGE LANGUAGE MODELS (LLMs)

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

Transformer-based LLMs are models that uses the transformer architecture and are pretrained on huge amounts of text data

Encoder-Only LLMs

Turns text into embeddings

Popular models:

- BERT – Bidirectional Encoder Representations from Transformers

A base LLM can have many variants:

- *RoBERTa – better performance*
- *DistilBERT – smaller and faster*
- *BERT-QA – fine-tuned for question answering*
- *BioBERT – fine-tuned for biomedical texts*
- *LegalBERT – fine-tuned for legal texts*

Decoder-Only LLMs

Infers the next token in a text

Popular models:

- GPT – Generative Pre-trained Transformer

Encoder-Decoder LLMs

Turns text into other text

Popular models:

- T5 – Text-to-Text Transfer Transformer
- BART – combines BERT and GPT

Once trained, **all the parameters are set** *(embeddings, attention weights, FNN weights, etc.)* so that anyone can input in a new text sequence and quickly calculate an output

TRAINING LLMs

Transformers & LLMs

Embeddings

Attention

FNNs

Encoders & Decoders

Pretrained LLMs

To **train an LLM**, researchers first choose an architecture, decide how they'll structure the inputs and outputs, and then select data sets for training

- Training is typically done on billions of words from a variety of sources
- This process is done by large tech companies and can take weeks or months to do on many GPUs (graphics processing units that allow for parallel processing)

Encoder-Only LLMs

Often trained with Masked Language Modeling (MLM)

- **Input:** "I love {MASK} lemonade"
- **Output:** "cold"

Decoder-Only LLMs

Often trained with Autoregressive Language Modeling

- **Input:** "I love cold"
- **Output:** "lemonade"

Encoder-Decoder LLMs

Often trained to handle a variety of tasks by converting everything to text

- **Input:** EN to ES: "I love lemonade"
- **Output:** "Me encanta la limonada"

POPULAR PRETRAINED LLMs

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

These are some of the most **popular pretrained LLMs**:

LLM	Type	Created By	Parameters	Trained On
BERT	Encoder-only	Google	110 million (BERT-base) 340 million (BERT-large)	BooksCorpus (800M words) English Wikipedia (2.5 billion words)
DistilBERT	Encoder-only	Hugging Face	66 million	BooksCorpus (800M words) English Wikipedia (2.5 billion words)
T5	Encoder-Decoder	Google	220 million (T5-small) 770 million (T5-large) 11 billion (T5-11B)	C4 (Colossal Clean Crawled Corpus) ~750GB of text
BART	Encoder-Decoder	Meta (Facebook)	140 million (BART-base) 400 million (BART-large)	CC-News (140GB of news articles) BooksCorpus (800M words) English Wikipedia (2.5 billion words)
GPT	Decoder-only	OpenAI	175 billion (GPT-3) 1.5 billion (GPT-2)	WebText (40GB of text from Reddit links) BooksCorpus (800M words) English Wikipedia (2.5 billion words)

Many of these are open source, but GPT-3 and later versions are proprietary – OpenAI hasn't even shared the number of parameters for GPT-4!

Each LLM can be used for **multiple applications**:

- BERT is used for sentiment analysis, classification, etc.
- T5 is used for summarization, translation, etc.
- GPT is mainly used for generation
- Any of them can be used for question answering, etc.

EXERCISE: TRANSFORMERS & LLMs

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

Match each layer with its definition:

Attention	
Embeddings	Adjusts the meaning of words based on surrounding words
FNN	Learns patterns from the prior layers and adds complexity

Provides numeric representations of text

Match the transformer type to its definition, application, and popular LLM:

BERT	Generates text	GPT	Text classification	Text generation
Translation	T5	Understands text	Understands & generates text	

	Definition	Application	LLM
Encoder-Only			
Decoder-Only			
Encoder-Decoder			

EXERCISE: TRANSFORMERS & LLMs

- Transformers & LLMs
- Embeddings
- Attention
- FNNs
- Encoders & Decoders
- Pretrained LLMs

Answer these questions:

What are transformers?

ANSWER BELOW

What is the most important part of the transformer architecture?

ANSWER BELOW

What are LLMs?

ANSWER BELOW

What are the main companies that are creating these LLMs?

ANSWER BELOW

Where do they overlap?

ANSWER BELOW

What does it mean to use a pretrained LLM?

ANSWER BELOW

KEY TAKEAWAYS

Transformers are a deep learning architecture with three main layers

- *The embeddings layer contains numeric vectors that represent semantic meaning of words*
- *The attention layer adjusts the meaning of each word based on context from surrounding words*
- *The feedforward neural network layer learns patterns from the prior layers and adds complexity*

Attention is the game changer in the transformer architecture

- *It retains context by using attention scores to capture relationships between words*
- *Matrix calculations allow for parallelization, making it possible to train on huge datasets*

Transformer-based LLMs are the most popular approach to NLP tasks

- *Large Language Models (LLMs) are deep learning models pretrained on huge text datasets*
- *Popular LLMs include encoder-only BERT for understanding text, decoder-only GPT for generating text and encoder-decoders T5 and BART for understanding and converting text*

HUGGING FACE TRANSFORMERS

HUGGING FACE TRANSFORMERS

In this section, we'll introduce the **Hugging Face Transformers** library in Python and walk through examples of how you can use pretrained models to perform NLP tasks

TOPICS WE'LL COVER:

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

GOALS FOR THIS SECTION:

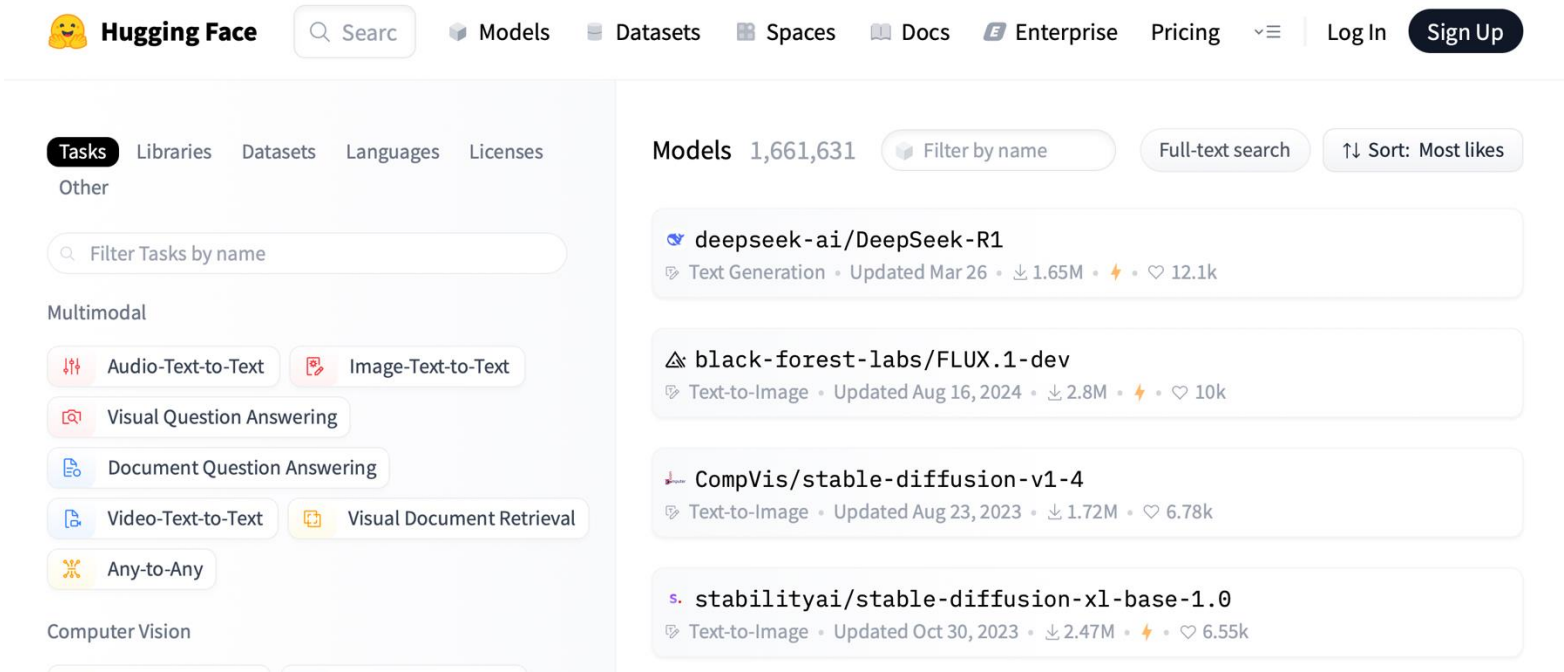
- Become familiar with the Hugging Face syntax and workflow
- Practice applying several types of NLP tasks using Hugging Face's pretrained models

HUGGING FACE

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

Hugging Face is the company that created the Transformers Python library, making it easy for data professionals to access and utilize pretrained LLMs

- They also host the Model Hub, which contains 1M+ pretrained, open-source models (in addition to base models, there are variants, fine-tuned models, experimental models, etc.)



HUGGING FACE WORKFLOW

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

We'll be using this **Hugging Face workflow** in Python for multiple applications across the three main LLM categories, as well as embeddings:

1 Determine your goal

Encoder-Only

- Sentiment Analysis
- Named Entity Recognition

Decoder-Only

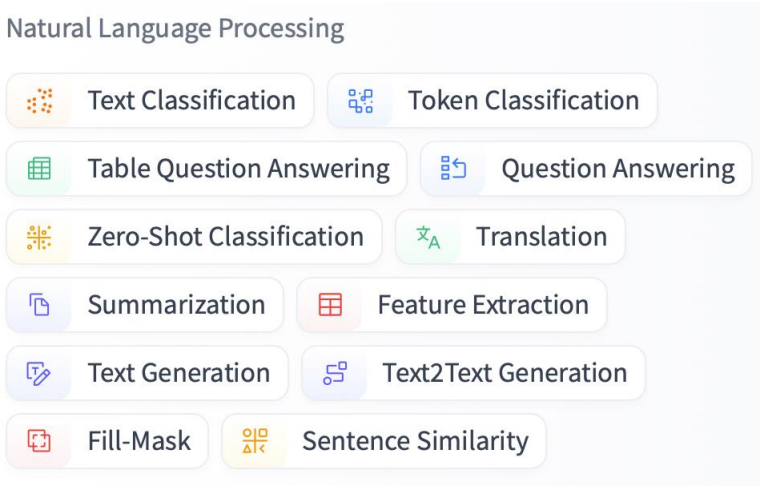
- Text Generation

Encoder-Decoder

- Zero-Shot Classification
- Text Summarization

Embedding

- Document Similarity

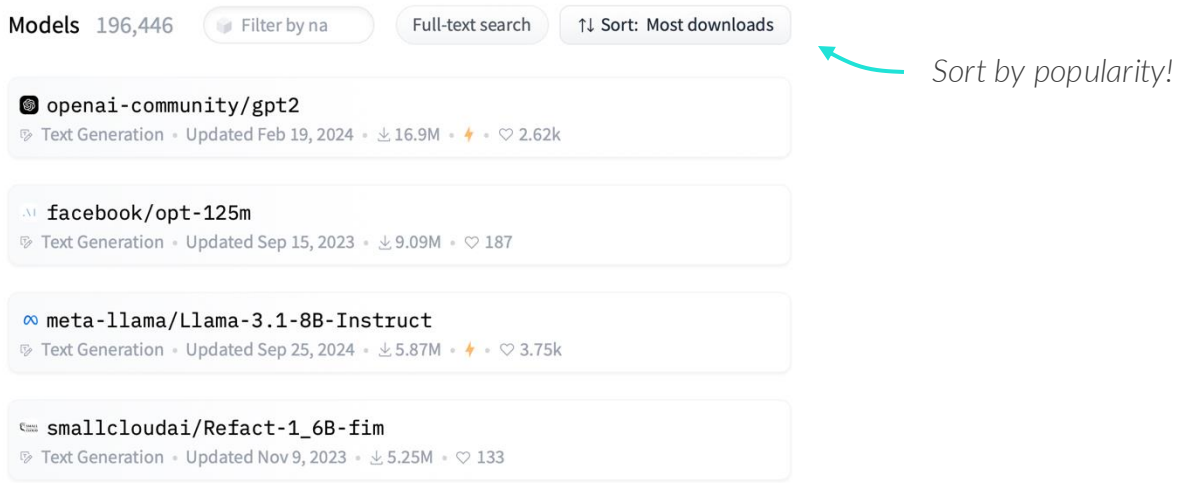


HUGGING FACE WORKFLOW

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

We'll be using this **Hugging Face workflow** in Python for multiple applications across the three main LLM categories, as well as embeddings:

1. **Determine your goal** *(Sentiment analysis, summarization, generation, etc.)*
2. **Identify a pretrained model** from Hugging Face's Model Hub



Sort by popularity!

HUGGING FACE WORKFLOW

We'll be using this **Hugging Face workflow** in Python for multiple applications across the three main LLM categories, as well as embeddings:

1. **Determine your goal** *(Sentiment analysis, summarization, generation, etc.)*
2. **Identify a pretrained model** from Hugging Face's Model Hub
3. **Specify your input data** *(a single string, a Series or column of text data, etc.)*
4. **Apply the pretrained model** on your input data and view the outputs

After using a pretrained model, you have the **optional step** of improving your results using transfer learning, fine-tuning, RAGs, and more (reference the Pretrained Deep Learning Models lesson), but those often require large labeled data sets and additional processing power

SENTIMENT ANALYSIS

Sentiment analysis is used to determine the positivity or negativity of text

- The default LLM for sentiment analysis is **DistilBERT** *(encoder-only)*

```
from transformers import pipeline

sentiment_analyzer = pipeline("sentiment-analysis",
                              model="distilbert/distilbert-base-uncased-finetuned-sst-2-english",
                              device=-1)

text = 'When life gives you lemons, make lemonade! 🙂'

sentiment_analyzer(text)
```

```
[{'label': 'POSITIVE', 'score': 0.996239423751831}]
```

*The predicted sentiment is **positive** (vs. negative)*

This is the model's confidence in its prediction (from 0 to 1)

Transformer pipeline steps:

1. *Import the pipeline module*
2. *Specify the task: **sentiment-analysis***
3. *Choose the default model*
4. *Specify we're only using our CPU*

This is very much a positive sentence

SENTIMENT ANALYSIS

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

EXAMPLE

Find the sentiment for Pop Chip reviews

Add a timer to compare performance

Hide warning messages

Speed up the code by using GPU instead of CPU

```
%%time
# add a timer

from transformers import pipeline, logging

# hide all non-critical warnings
logging.set_verbosity_error()

# create a sentiment analyzer
sentiment_analyzer = pipeline("sentiment-analysis",
                              model="distilbert/distilbert-base-uncased-finetuned-sst-2-english",
                              device="mps", # speed up the query by using our GPU if available
                              truncation=True)

# apply the sentiment analyzer to a series
sentiment_scores = df.Text.apply(sentiment_analyzer)
sentiment_scores
```

```
CPU times: user 1.58 s, sys: 988 ms, total: 2.57 s
Wall time: 1.65 s
0    [{'label': 'POSITIVE', 'score': 0.9935213923454285}]
1     [{'label': 'POSITIVE', 'score': 0.999605119228363}]
2    [{'label': 'NEGATIVE', 'score': 0.6984829306602478}]
3    [{'label': 'NEGATIVE', 'score': 0.9996308088302612}]
4    [{'label': 'POSITIVE', 'score': 0.9991814494132996}]
Name: Text, dtype: object
```

Use .apply to apply sentiment_analyzer to a column of truncated text

PRO TIP: SPEEDING UP TRANSFORMERS CODE

Here are a few tips for **speeding up** your transformers code:

- For new Macs with an Apple Silicon chip *(i.e., M3 chip)*, choose **device='mps'** to use the GPU
- For PCs with a dedicated GPU *(i.e., gaming PCs)*, choose **device='cuda'** to use your GPU
- For older Macs and most PCs, you'll only be able to use your CPU for computations *(default)*

```
%%time

from transformers import pipeline

sentiment_analyzer = pipeline(
    "sentiment-analysis",
    model="distilbert-base-uncased-finetuned-sst-2-english", # 1. smaller model
    device=-1, # running on CPU
    truncation=True,
    use_fast=True # 2. faster tokenization
)

import torch
torch.set_num_threads(1)  # 3. specify multi-threading

with torch.no_grad(): # 4. disable gradients
    sentiment_scores = df['Text'].apply(sentiment_analyzer)
```

While not as good as using a GPU, you can try some of these techniques to speed up your code if you only have a CPU available

I was able to cut my runtime from 1 minute to 15 seconds by trying these 4 things!

ASSIGNMENT: SENTIMENT ANALYSIS WITH LLMs

NEW MESSAGE

May 28, 2025

From: **Oscar Wynn** *(The Movie Maven)*

Subject: **RE: Feel good vs dark movies**

Regarding my earlier message, can you do this using Hugging Face & LLMs instead of VADER & rules, and compare the results? Thank you!

We're publishing an article on the top 10 most feel-good movies and the top 10 darkest movies according to data.

Could you use sentiment analysis to help us come up with movies for these two lists?

Thanks!
Oscar

movie_reviews_sentiment.csv

Reply Forward

Key Objectives

1. Create a new "nlp_transformers" environment
2. Launch Jupyter Notebook
3. Read in the movie reviews data set including the VADER sentiment scores
4. Apply sentiment analysis to the "movie_info" column using transformers
5. Compare the transformers sentiment scores with the VADER sentiment scores

NAMED ENTITY RECOGNITION (NER)

Named Entity Recognition (NER) is used to find and label important information *(people, places, organizations, dates, etc.)* in text

- The default LLM for NER is **BERT** *(encoder-only)*

```
from transformers import pipeline

ner_analyzer = pipeline("ner",
                        model='dbmdz/bert-large-cased-finetuned-conll03-english',
                        device=-1,
                        aggregation_strategy='SIMPLE')

text = "I ordered an Arnold Palmer at Applebee's in Springfield."

ner_analyzer(text)
```

Transformer pipeline steps:

1. *Import the pipeline module*
2. *Specify the task:* ***ner***
3. *Choose the default model*
4. *Specify we're only using our CPU*

```
[{'entity_group': 'MISC',
  'score': np.float32(0.9914088),
  'word': 'Arnold Palmer',
  'start': 13,
  'end': 26},
 {'entity_group': 'ORG',
  'score': np.float32(0.9436141),
  'word': "Applebee ' s",
  'start': 30,
  'end': 40},
 {'entity_group': 'LOC',
  'score': np.float32(0.9780036),
  'word': 'Springfield',
  'start': 44,
  'end': 55}]
```

By setting aggregation_strategy to "SIMPLE", we're specifying we want to look at words, not subwords

We're 97.8% sure "Springfield" is a named entity identified as a location (LOC) – the word starts at character 44 and ends at 55

NAMED ENTITY RECOGNITION (NER)

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

EXAMPLE *Find common terms mentioned in Pop Chip reviews*

```
# find the named entities in each review
ner_analyzer = pipeline("ner",
                        model="dbmdz/bert-large-cased-finetuned-conll03-english",
                        device='mps',
                        aggregation_strategy='SIMPLE')

# apply to all reviews
df['Named_Entities'] = df['Text'].apply(lambda text:
                                        [entity['word'] for entity in ner_analyzer(text)])

# create a unique list of named entities
named_entities = list(set(df.Named_Entities.explode().dropna().tolist()))

# exclude subwords from the list
named_entities_clean = [entity for entity in named_entities if '#' not in entity]
sorted(named_entities_clean)
```

Use .apply to apply ner_analyzer to a column of text, create a list of named entities, and clean up the list

```
['& V',
 'Amazon',
 'B',
 'COSTCO',
 'Ch',
 'Cheetos',
 'Chip',
 'Costco',
 'General Mills',
 'Lays',
 'Miami',
 'PopChips',
 'Popchi',
 'Popchips',
 'Popchips B',
 'Pringles',
 'S',
 'S & V',
 'Salt',
 'Salt and Vinegar',
 'Stop and Shop',
 'VA',
 "Vinegar Pirate ' s Bo",
 'Watch',
 'and',
 'com']
```

You can see competitors in the output list

ASSIGNMENT: NAMED ENTITY RECOGNITION (NER)

NEW MESSAGE

May 29, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **Book characters**

Hi!

It's been a while.

Our client would like a rough list of characters from our book collection.

Could you use NER to extract the named entities from the book descriptions, and then filter on only people?

Thanks so much!
Lexi

childrens_books.csv

Reply | Forward

Key Objectives

1. Read in the children's books data set
2. Apply NER to the Description column
3. Create a list of all named entities
4. Only include the people (PER)
5. *Extra credit:* Exclude the authors as well

ZERO-SHOT CLASSIFICATION

Zero-Shot classification is used to quickly categorize text without labels

- The default LLM for zero-shot classification is **BART** *(encoder-decoder)*

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli",
                      device=-1)

text = 'When life gives you lemons, make lemonade! 🙂'

classifier(text, ['quote', 'food & drinks', 'technology'])
```

*Same pipeline steps with **zero-shot-classification** as the task*

```
{'sequence': 'When life gives you lemons, make lemonade! 🙂',
 'labels': ['quote', 'food & drinks', 'technology'],
 'scores': [0.9833195209503174, 0.011176466010510921, 0.005504013504832983]}
```

This is a quote!

You provide the label options and the model returns scores that classify it into one of those labels (adding up to 1)

ZERO-SHOT CLASSIFICATION

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

EXAMPLE

Categorize pop chip reviews into one of five groups

```
# remember our topics from the ml section: order, taste & texture, good, flavor, health
classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli",
                      device='mps')

# apply to all reviews
df['Category'] = (df['Text']
                  .apply(lambda x:
                         classifier(x, candidate_labels=['order', 'taste & texture', 'good',
                                                         'flavor', 'health'])['labels'][0]))
# view the labels
df[['Text', 'Category']].head()
```

Use things like domain expertise, EDA, and topic modeling to come up with relevant labels

	Text	Category
0	Popchips are the bomb!! I use the parmesan garlic to scoop up cottage cheese as a healthy alternative to chips and dip. My healthy eating program is saved.	health
1	I like the puffed nature of this chip that makes it more unique in the chip market. I ordered the Salt and Vinegar and absolutely love that flavor, hands down my favorite chip ever. I have tried the cheddar and regular flavors as well. The cheddar is about a 4/5 and the regular is about a 3/5 because I prefer strong flavors and obviously that would not be the case for the regular. The Salt and Vinegar is kind of weak compared to some regular S&V chips, but is quite flavorful and makes you wanting to come back for more.	flavor
2	I just love these chips! I was always a big fan of potato chips, but haven't had one since I discovered popchips. They are great for dipping or all alone. I am constantly re-ordering them. One note however-if you are on a low salt diet these chips are probably not for you. They are high in sodium. We go through a case every two months. If you love them it pays to join the subscribe and save program through Amazon. You save money and stay supplied!	good
3	These tasted like potatoe stix, that we got in grade school with our lunches usually on pizza day. They were the bomb then, not so much now. Won't buy again unless I get them for cheap or free.	taste & texture

It looks like these labels make sense!

ASSIGNMENT: ZERO-SHOT CLASSIFICATION

NEW MESSAGE

May 30, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **Book categories**

Hello,

Our client would like to divide their book list into five shelves at their physical bookstore. Could you label all the books as one of these categories?

- Adventure & Fantasy
- Animals & Nature
- Mystery
- Humor
- Non-Fiction

Thanks!
Lexi

Reply Forward

Key Objectives

1. Apply zero-shot classification to the Description column
2. Find the number of books in each category and check a few to see if the results make sense

TEXT SUMMARIZATION

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

Text summarization is used to make long bodies of text more concise

- The default LLM for text summarization is **BART** *(encoder-decoder)*

```
from transformers import pipeline

summarizer = pipeline("summarization",
                      model="facebook/bart-large-cnn",
                      device=-1)

text = """The lemon tree produces a pointed oval yellow fruit. Botanically this is a hesperidium, a modified berry with
a tough, leathery rind. The rind is divided into an outer colored layer or zest, which is aromatic with essential oils,
and an inner layer of white spongy pith. Inside are multiple carpels arranged as radial segments. The seeds develop inside
the carpels. The space inside each segment is a locule filled with juice vesicles. Lemons contain many phytochemicals,
including polyphenols, terpenes, and tannins.[3] Their juice contains slightly more citric acid than lime juice (about
47 g/L), nearly twice as much as grapefruit juice, and about five times as much as orange juice.[4]"""

summarizer(text, min_length=20, max_length=50)
```

*Same pipeline steps with **summarization** as the task*

```
'The lemon tree produces a pointed oval yellow fruit. Lemons contain many phytochemicals, including
 polyphenols, terpenes, and tannins. Their juice contains slightly more citric acid than lime juice.'
```

Beyond specifying the min and max length of the summarized text, you can **set the do_sample parameter** to False to use the most likely next word (default) or to True to use a more random and creative next word

TEXT SUMMARIZATION

EXAMPLE

Use text summarization to reduce text size before sentiment analysis

```
# load pipelines
summarizer = pipeline("summarization", model="facebook/bart-large-cnn", device='mps')
sentiment_analyzer = pipeline("sentiment-analysis",
                              model="distilbert-base-uncased-finetuned-sst-2-english",
                              device='mps')

# step 1: summarize reviews
df['Summary_HF'] = df['Text'].apply(lambda x:
                                    summarizer(x, max_length=100)[0]['summary_text'])

# step 2: find sentiment scores
sentiment_scores2 = df['Summary_HF'].apply(sentiment_analyzer)

# extract label and score and create a sentiment score
df['Label_HF2'] = sentiment_scores2.apply(lambda x: x[0]['label'])
df['Score_HF2'] = sentiment_scores2.apply(lambda x: x[0]['score'])
df['Sentiment_HF2'] = df.apply(lambda row: row['Score_HF2']
                               if row['Label_HF2'] == 'POSITIVE' else -row['Score_HF2'], axis=1)

# compare the sentiment scores
df[['Text', 'Sentiment_HF', 'Sentiment_HF2']].head()
```

Earlier we truncated the data because our text was too long for sentiment analysis, but we can apply text summarization instead

Text	Sentiment_VADER	Sentiment_HF	Sentiment_HF2
I just love these chips! I was always a big fan of potato chips, but haven't had one since I discovered popchips. They are great for dipping or all alone. I am constantly re-ordering them. One note however-if you are on a low salt diet these chips are probably not for you. They are high in sodium. We go through a case every two months. If you love them it pays to join the subscribe and save program through Amazon. You save money and stay supplied!	0.9790	-0.698483	-0.992971

None of these 3 perfectly capture the sentiment

ASSIGNMENT: TEXT SUMMARIZATION

NEW MESSAGE

May 31, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **Book summaries**

Hello,

Our client would like a short one-liner for each book. Could you use text summarization to summarize the descriptions?

Thanks!
Lexi

Reply Forward

Key Objectives

1. Apply text summarization to the Description column
2. Review the results to see if they make sense

PRO TIP: TEXT GENERATION

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

Text generation is used to write new text based on a prompt

- The default LLM for text generation is **GPT** *(decoder-only)*

```
from transformers import pipeline

generator = pipeline("text-generation", model="gpt2", device=-1)

prompt = "On a hot summer day, I love to drink cold lemonade because"
```

```
generator(prompt, max_length=50, num_return_sequences=1, do_sample=False)
```

```
[{'generated_text': "On a hot summer day, I love to drink cold lemonade because it's so refreshing. I also love to drink cold lemonade becaus
e it's so refreshing.\n\nI love to drink cold lemonade because it's so refreshing. I also"}]
```

```
generator(prompt, max_length=50, num_return_sequences=1, do_sample=True)
```

```
[{'generated_text': "On a hot summer day, I love to drink cold lemonade because there really are no other options. There's no way it'll keep
me warm and dry, if it does end up freezing, it won't be good for you. It's"}]
```

```
generator(prompt, max_length=50, num_return_sequences=1, do_sample=True)
```

```
[{'generated_text': "On a hot summer day, I love to drink cold lemonade because I am always having a good time doing it! I've found it is eas
ier to drink lemonade without actually making any lemonade.\n\nThis hot summer day I was inspired"}]
```

The do_sample parameter allows you to get more random and creative next words

Text generation is mostly used for **creating applications**, and better models like GPT-3 and 4 require using an API with an OpenAI account and credits

DOCUMENT EMBEDDINGS

- Hugging Face Overview
- Sentiment Analysis
- Named Entity Recognition
- Zero-Shot Classification
- Text Summarization
- Text Generation
- Document Similarity

Embeddings are numeric representations of text that carry semantic meaning

- You can get document embeddings from an LLM using feature extraction
- The default LLM for feature extraction is **BERT** *(encoder-only)*, but **MiniLM** *(encoder-only)* is more popular for document similarity

```
from transformers import pipeline

feature_extractor = pipeline("feature-extraction",
                             model="sentence-transformers/all-MiniLM-L6-v2",
                             device=-1)

text = 'When life gives you lemons, make lemonade! 🙂'
```

*Same pipeline steps with **feature-extraction** as the task, and a non-default, popular model*

```
# view the embedding
feature_extractor(text)[0][0][:5]
```

```
[-0.2936323285102844,
 0.20775198936462402,
 0.11103478074073792,
 0.14668866991996765,
 0.39885425567626953]
```

Now that the sentence has been vectorized, you can apply EDA, clustering, classification, etc.

```
# view the shape
len(feature_extractor(text)[0][0])
```

```
384
```

MiniLM uses 384 dimensions for embeddings compared to the 768 dimensions from BERT

Feature extraction is the idea of using embeddings from (typically) the last layer of a pretrained transformer model and inputting them into downstream ML / analysis tasks

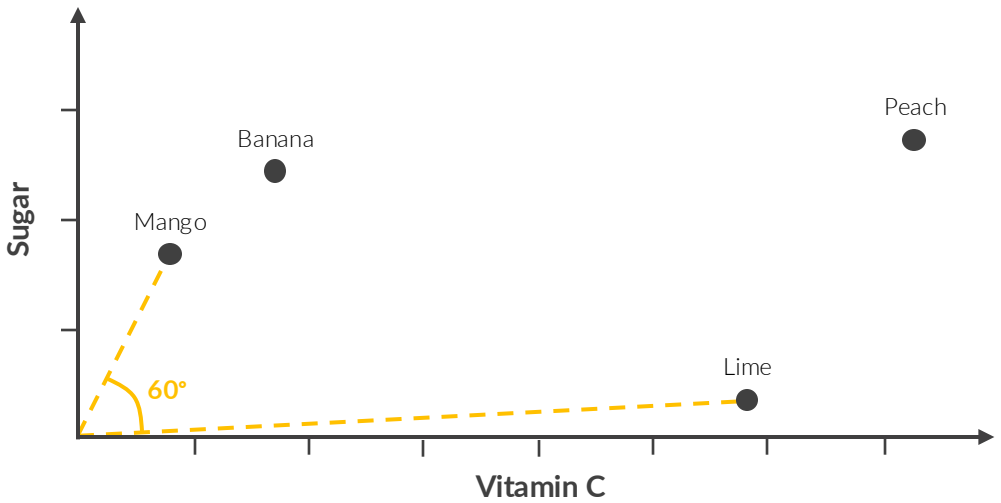
COSINE SIMILARITY

Cosine similarity is a metric used to calculate the similarity between observations

- Values range from -1 (dissimilar) to +1 (similar)
- This can be used with document embeddings to calculate **document similarity**

EXAMPLE

Finding the fruit most similar to mango



$$\cos(60) = 0.5$$

Mangos and limes are not very similar

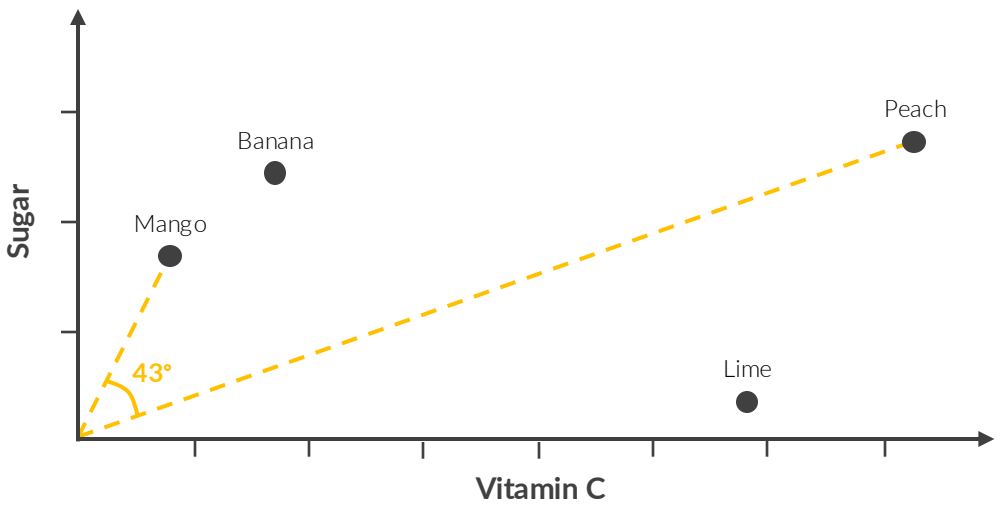
COSINE SIMILARITY

Cosine similarity is a metric used to calculate the similarity between observations

- Values range from -1 (dissimilar) to +1 (similar)
- This can be used with document embeddings to calculate **document similarity**

EXAMPLE

Finding the fruit most similar to mango



$$\cos(43) = 0.73$$

Peaches are more similar to mangos than limes

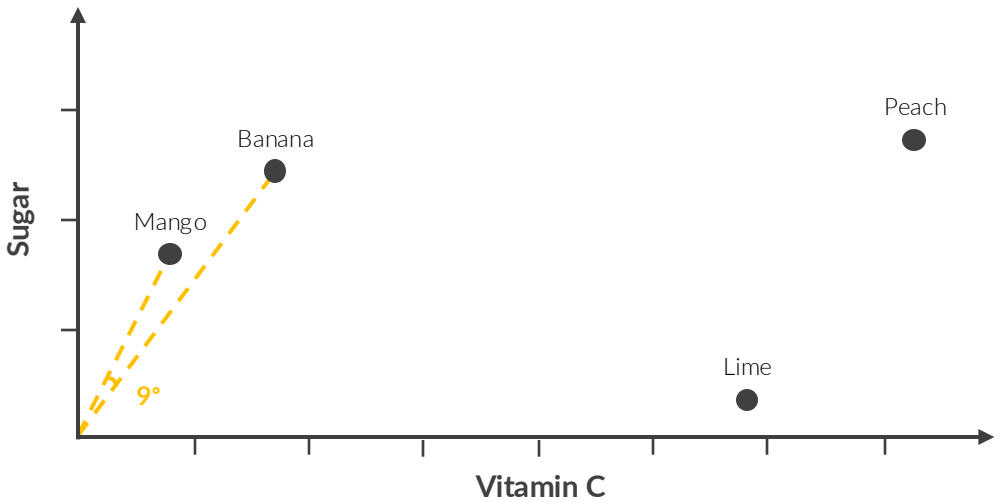
COSINE SIMILARITY

Cosine similarity is a metric used to calculate the similarity between observations

- Values range from -1 (dissimilar) to +1 (similar)
- This can be used with document embeddings to calculate **document similarity**

EXAMPLE

Finding the fruit most similar to mango



$$\cos(9) = 0.98$$

Bananas are the most similar fruit to mangos!

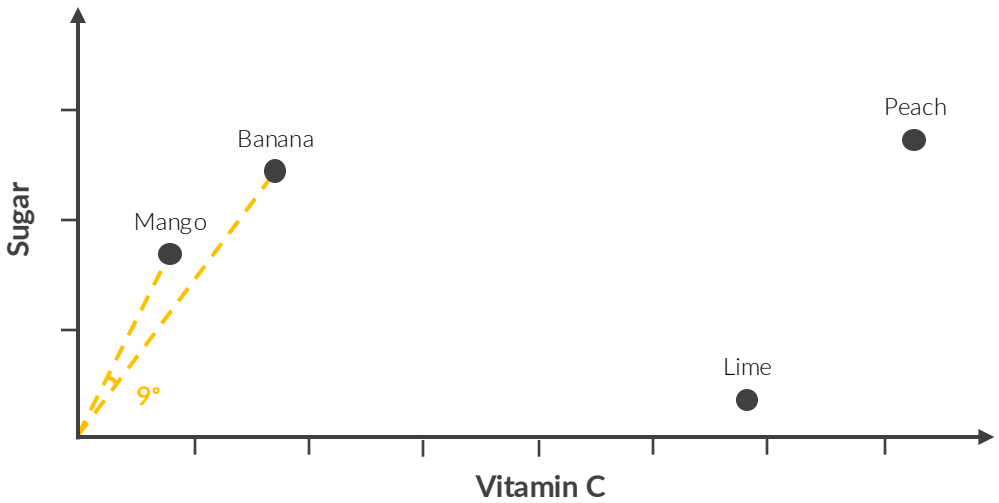
COSINE SIMILARITY

Cosine similarity is a metric used to calculate the similarity between observations

- Values range from -1 (dissimilar) to +1 (similar)
- This can be used with document embeddings to calculate **document similarity**

EXAMPLE

Finding the fruit most similar to mango



There are many similarity metrics to choose from, but cosine similarity is popular in machine learning because:

- It focuses on **direction** instead of magnitude
- It can handle **high dimensions**
- It works well on **sparse data** (data containing many 0 values)

DOCUMENT SIMILARITY

EXAMPLE

Use embeddings and cosine similarity to find similar movies

```
from transformers import pipeline
import numpy as np

# specify where we're getting our embeddings from
feature_extractor = pipeline("feature-extraction",
                             model="sentence-transformers/all-MiniLM-L6-v2",
                             device='mps')

# extract the embeddings
embeddings = movies['movie_info'].apply(lambda x: feature_extractor(x)[0][0])
movie_embeddings = np.vstack(embeddings)
movie_embeddings[0][:10]
```

```
array([-0.32529733, -0.07725151,  0.12645775, -0.07083085, -0.19548225,
        0.39315301, -0.02867479, -0.0734244 , -0.04998297, -0.22656319])
```

```
# number of movies
len(movie_embeddings)
```

```
166
```

```
# size of each embedding
len(movie_embeddings[0])
```

```
384
```

In layman's terms, movie_embeddings holds the embedding for each movie

In technical terms, we're creating a numpy array with:

- *166 elements (one for each movie)*
- *384 dimensions (movie vector)*

DOCUMENT SIMILARITY

EXAMPLE

Use embeddings and cosine similarity to find similar movies

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# create a function to get movie recommendations
def get_similar_movies(embeddings, movie_index, movie_details, top_n=5):

    # use cosine similarity: more technical details in the demo!
```

```
# find movies similar to Captain Marvel
get_similar_movies(embeddings_movies, 25, movies[['movie_title', 'movie_info']])
```

	Movie	Similarity_Score
25	Captain Marvel	1.000000
45	Fast & Furious Presents: Hobbs & Shaw	0.819661
18	Avengers: Endgame	0.794008
131	The LEGO Movie 2: The Second Part	0.792253
6	Alita: Battle Angel	0.789453
31	Dark Phoenix	0.787654

These are the movies that are most similar to Captain Marvel based on their movie descriptions

ASSIGNMENT: DOCUMENT SIMILARITY

NEW MESSAGE

June 1, 2025

From: **Lexi Con** *(Lead Data Scientist)*

Subject: **Book recommendations**

Hello,

I have one final request for you.

Our client is a big fan of the first Harry Potter book, Harry Potter and the Sorcerer's Stone.

What other books would you recommend for them using document similarity with LLM embeddings?

Thanks for all your help over the past few weeks!
Lexi

Reply Forward

Key Objectives

1. Turn the Description column into embeddings using feature extraction
2. Compare the cosine similarity of Harry Potter and the Sorcerer's Stone compared to all other books
3. Return the top 5 most similar books

KEY TAKEAWAYS

The **transformers library** allows you to use pretrained LLMs in Python

- *Most data professionals will use pretrained models, while researchers and specialists will fine-tune models*
- *The steps are to select a task, specify a model, apply the model on your text data, and view the output*
- *The code can take a long time to run so be sure to utilize your GPU, if available (device = 'mps' or 'cuda')*

Hugging Face's **Model Hub** contains many NLP tasks to choose from

- *The transformers library will provide a default model for various tasks, but you can swap out models*
- *By filtering on tasks and sorting on downloads, you can find alternative models to test out*

There are many **applications** of LLMs

- *In this section, we covered sentiment analysis, named entity recognition, zero-shot classification, text summarization, text generation and document similarity*
- *LLMs are just one tool in a data scientist's toolbox – these techniques can be used as an alternative to traditional techniques (sentiment analysis) or alongside traditional techniques (cosine similarity)*

NLP REVIEW & NEXT STEPS

NLP REVIEW

These are the **NLP techniques & applications** that we covered:

NLP Review

NLP Next Steps

NLP Category	Technique	Application
Traditional	Rules-Based	→ *Sentiment Analysis*
	Supervised Learning (Naïve Bayes)	→ *Text Classification*
	Unsupervised Learning (NMF)	→ *Topic Modeling*
Modern	Encoder-Only LLM (BERT)	→ *Sentiment Analysis* *Named Entity Recognition (NER)*
	Encoder-Decoder LLM (BART)	→ *Zero Shot Classification* *Text Summarization*
	Decoder-Only LLM (GPT)	→ *Text Generation*
	Embeddings (MiniLM)	→ *Document Similarity*

NLP REVIEW

NLP Review

NLP Next Steps

When should I use traditional vs. modern NLP techniques?

- In summary, start simple!

What is my NLP goal?

Sentiment Analysis

Text Classification

Topic Modeling

→ *These can be done with traditional techniques* →

How much data do I have?

Small to medium data (<100k rows) → ***Try traditional techniques first***

Big data (>1M rows) → ***Consider modern techniques***

Text Generation

Machine Translation

Question Answering

→ *These cannot be done with traditional techniques, so **use modern techniques***

NLP NEXT STEPS

If you enjoyed the technical aspects of this course and **want to learn more**:

NLP Review

NLP Next Steps

Traditional NLP

You now know the basics of text preprocessing and vectorization, and have practice applying supervised & unsupervised learning techniques

Next steps:

- **Start small:** Use these NLP techniques on a small piece of a larger data science project
- **Go beyond the basics:** Try practicing on text and numeric data, testing out different cleaning techniques, tuning model parameters, and mixing & matching techniques
- **Learn new algos:** You are well-equipped to learn other new machine learning techniques you encounter

Modern NLP

You now know the basics of neural networks, deep learning, and the transformer architecture, and have practice applying pretrained LLMs

Next steps (beyond pretrained models):

- **Transfer learning:** LLM parameters stay mostly frozen, and only the last layers are fine-tuned on a new task (useful if there is limited training data)
- **Fine-tuning:** All LLM parameters are updated based on new training data (requires 10k+ labeled examples)
- **RAGs (retrieval-augmented generation):** Enhances LLM outputs by retrieving relevant info from a database before generating answers (requires 10k+ external text documents)

NLP NEXT STEPS

NLP Review

NLP Next Steps

Key takeaways from someone who has been a decade-long data scientist:

Modern NLP is a **huge mindset shift** from traditional data science

- With traditional science, the "danger zone" lies in not understanding everything
- With modern NLP, it's impossible to comprehend everything

The **good** news:

- It's incredible how well modern NLP techniques work
- Powerful NLP techniques are now available to the masses

The **bad** news:

- AI ethics implications *(shout out to Chris Bruehl's Data & AI Ethics course!)*
- Sometimes it feels impossible to keep up

NLP NEXT STEPS

NLP Review

NLP Next Steps

After this course:

- You can follow the NLP conversation now that you understand the foundations
- There are a lot of advancements so keep track of the news!

Top organizations and labs:

- OpenAI (@OpenAI) – creator of GPT models
- Google AI (@GoogleAI) – creator of BERT, T5 and more
- Meta AI (@MetaAI) – creator of BART, LLaMA and more
- Hugging Face (@huggingface) – hosts NLP models
- Stanford NLP Group (@stanfordnlp) – leader in linguistic research
- DeepMind (@DeepMind) – cutting-edge AI research

Congrats on completing this course!

The NLP field is rapidly evolving and there are always new things to learn. You now have the baseline knowledge to go out, explore, test out new models and dive deeper into whatever interests you.

Welcome to the field of NLP. It's an exciting time, and we are just getting started. 🚀 😊