Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (**Raise Hand**). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, ***be sure to mute yourself again***.

Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class. Releasing these recordings is strictly prohibited.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

SEAS 8520 – Lecture 7: NLP Tokenization & Embeddings

Walid Hassan, M.B.A, D.Eng.

Assignment & Discussion Board

- **Will grade / review this week.**

Intuition Behind TF-IDF

Term Frequency (TF):

•Measures how frequently a term (word) appears in a document.

•Intuition: The more times a word appears in a document, the more important it is in that document.

• TF = (Number of repetitions of word in a document) / (# of words in a document)

• TF is a local measure (i.e. within a document)

IDF:

- Measures how important a term is in the entire corpus.
- Intuition: A term that appears in many documents is less informative about any particular document.
- The logarithm is used to dampen the effect of terms that are very common across all documents. IDF is a Global measure (i.e. across documents)

Intuition Behind TF-IDF

TF-IDF:

- Combines TF and IDF to balance the frequency of a term in a specific document with how common the term is across the corpus.
- Intuition: A high TF-IDF score indicates that the term is very important in the document but not common in the corpus.
- Balancing Frequency and Rarity: Balances the need to recognize frequently used terms within a document while downplaying those common across many documents.
- If a term appears in all documents, it is assumed to be not as 'informative' as a term that appears in only one document.

Data Preparation…

- **Cleaning**: Removing irrelevant characters such as punctuation, special characters, or numbers that may not be useful for certain analyses.
- **Dealing with Biases**: Ensuring the data does not reinforce or introduce biases that could affect the model's fairness and objectivity.
- **Normalization**: Converting text to a uniform case (upper or lower), which is important for consistency, especially in case-sensitive languages.
- **Handling Missing Values**: Identifying and imputing or removing missing values in the dataset to prevent errors during processing.

Libraries to Help with Data Prep…

Pandas: Essential for handling and manipulating structured data. Can be used for cleaning, transforming, and preparing datasets.

Scikit-learn: Offers tools for text feature extraction, such as CountVectorizer and TfidfVectorizer, which include options for tokenization and preprocessing.

NLTK (Natural Language Toolkit) :Provides functions for tokenization, part-of-speech tagging, and named entity recognition, among others.

spaCy: Offers robust features for text preprocessing, including sentence segmentation, part-of-speech tagging, and entity recognition, with support for multiple languages.

And many Others?

Thoughts?

- So we have Text
- Computers do not understand Text.
- Computers understand numbers.
- What can be done?

Text Standardization /Normalization

Consider these two sentences:

- “sunset came. i was staring at the Mexico sky. Isnt nature splendid??”
- “Sunset came; I stared at the México sky. Isn’t nature splendid?”

Src: Chollet, F. (2021). Deep Learning with Python, Second Edition (2nd ed.). Manning

Text Standardization /Normalization

- “sunset came. i was staring at the Mexico sky. Isnt nature splendid??”
- “Sunset came; I stared at the México sky. Isn’t nature splendid?”

Observations:

“i” and “I” are two different characters, “Mexico” and “México” are two different words,

“isnt” isn’t “isn’t,”….etc.

Text standardization is a basic form of feature engineering that aims to erase encoding differences that you don’t want your model to have to deal with.

Src: Chollet, F. (2021). Deep Learning with Python, Second Edition (2nd ed.). Manning

Text Standardization /Normalization

One of the simplest and most widespread standardization schemes is “convert to lowercase and remove punctuation characters.” Our two sentences would become

- “sunset came i was staring at the mexico sky isnt nature splendid”
- “sunset came i stared at the méxico sky isnt nature splendid”

Some other methods also remove special characters…etc.

Src: Chollet, F. (2021). Deep Learning with Python, Second Edition (2nd ed.). Manning

Tokenization…

Tokenization is the process of breaking down a text into smaller pieces called tokens. These tokens can represent words, sub-words, or even characters, depending on the granularity of the tokenization process. It helps in preparing the text for further processing, making it easier for algorithms to understand and manipulate..

But Characters are “ASCII/Unicode???”

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
https://en.wikipedia.org/wiki/Rule-based_machine_translation

One Idea is Character based…

Why not use “character based tokenization”…

It is much harder to understand the sentiment of the word…

i.e. We understand words not characters per se..

Src: Laurance Moroney, Zero to Hero NLP. https://www.youtube.com/watch?v=fNxaJsNG3-s&list=PPSV

Another idea is word based

This method utilizes inherent separators, primarily spaces, to define tokens. Although this seems reasonable, it presents certain limitations.

- Firstly, it fails to recognize the connection between words and their base forms. For instance, "eat" and "eating" are treated as distinct entities without any acknowledged link.
- Secondly, it results in an expansive vocabulary. Since space serves as the basis for generating new tokens, the token count escalates without any mechanism to cluster related words, leading to increased vocabulary size.

Another idea is word based

- Out-of-Vocabulary (OOV) Words: When a word that was not seen during training appears in the test data, full-word tokenization has no way of handling it other than treating it as an unknown token. This can impair the model's ability to generalize to new text.

- Lack of Morphological Awareness: Full-word tokenization does not capture the morphological relationships between words, such as the connection between "eat" and "eating" or between "happy" and "unhappy." This can limit the model's understanding of the semantic and syntactic nuances of language

Byte-Pair Encoding (BPE)

This method is a compromise between letters and full words is used, and the final Tokenization vocabulary includes both common words and word fragments from which larger and less frequent words can be composed. The vocabulary is computed using a sub-word tokenizer that greedily merges commonly occurring sub-strings based on their frequency.

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Byte-Pair Encoding (BPE)

Step a) A passage of text from a nursery rhyme. The tokens are initially just the characters and whitespace (represented by an underscore), and their frequencies are displayed in the table

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Byte-Pair Encoding (BPE)

Step b) At each iteration, the sub-word tokenizer looks for the most commonly occurring adjacent pair of tokens (in this case, se) and merges them

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Byte-Pair Encoding (BPE)

Step C) At the second iteration, the algorithm merges e and the whitespace character_. Note that the last character of the first
token to be merged cannot be whitespace, which prevents merging across words.

c) a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Byte-Pair Encoding (BPE)

d)

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

Step d) After 22 iterations, the tokens consist of a mix of letters, word fragments, and commonly occurring words.

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Byte-Pair Encoding (BPE)

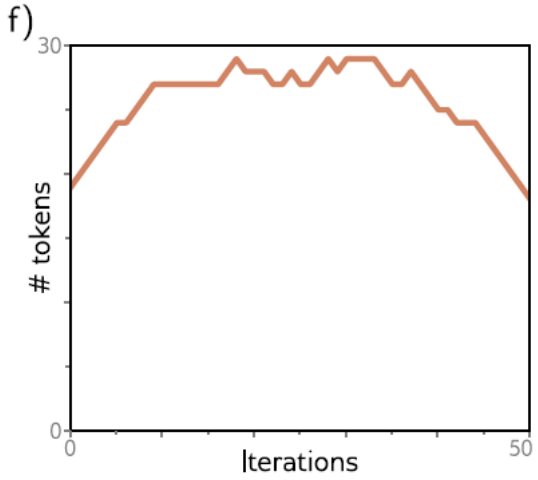
e)

see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

Step e) If we continue this process indefinitely, the tokens eventually represent the full words.

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

Byte-Pair Encoding (BPE)



Over time, the number of tokens increases as we add word fragments to the letters and then decreases again as we merge these fragments. In a real situation, there would be a very large number of words, and the algorithm would terminate when the vocabulary size (number of tokens) reached a predetermined value. Punctuation and capital letters would also be treated as separate input characters

Src: Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.

BPE Benefits

Roots and Affixes: By splitting words into subwords, BPE can capture the root of a word and its affixes separately. For example, "eating" might be split into "eat" and "ing", where "eat" is the root, and "ing" is a suffix indicating continuous tense. This allows the model to recognize that "eat" and "eating" share a common root, helping it to generalize across word forms.

Reducing Vocabulary Size: Since BPE creates a vocabulary that includes common sub-words (or byte pairs) rather than every possible word form, it can significantly reduce the size of the vocabulary needed to cover the training corpus. This reduction in vocabulary size decreases memory requirements and can improve the efficiency of the model.

BPE Benefits

Handling OOV Words: With sub-word tokenization, even if the model encounters a word it has never seen before, it can still process the word by breaking it down into known sub-words, allowing for better handling of OOV words and improving the model's ability to generalize.

Morphological Awareness: Sub-word tokenization can capture some level of morphological information, which can be particularly beneficial for languages with rich morphology or when dealing with inflected forms of words.

WordPiece

WordPiece also starts with individual characters and iteratively merges symbols to form sub-words. However, it introduces a slightly different criterion for merging. Instead of just frequency, WordPiece considers the likelihood of the new symbol when added to the model's existing vocabulary.

The objective is to maximize the likelihood of the training data given the current vocabulary, which includes choosing merges that improve the model's performance on the dataset. This approach can lead to a vocabulary that is more optimized for the specific dataset or task.

i.e. WordPiece becomes a part of the larger model, not just tokenization.

Example: BERT.

SentencePiece

This tokenization method is capable of handling languages without clear word boundaries or those that benefit from a tokenizer that operates directly on raw text without assuming pre-tokenization into words.

This is specifically needed for languages where spaces is not required to separate words (e.g. Japanese….etc).

SentencePiece

SOV : Japanese (subject)-(object)-(verb)	Literal translation following the word order
私は本を読みます。 Watashi wa hon o yomimasu.	I / the book / read.
私は音楽家です。 Watashi wa ongakuka desu.	I / a musician / am.
その本は音楽についてです。 Sono hon wa ongaku ni tsuite desu.	The book / the music / about / is.

The structure is different from Subject Verb Object and there is not separation of words!

Src: https://www.japanesepod101.com/blog/2020/08/07/japanese-word-order/

Tokenization…

Let us use NLTK tokenizer as an example…

sentence = "hello class, welcome to the world of Natural Language Processing“

Tokens:

['hello', 'class', ',', 'welcome', 'to', 'the', 'world', 'of', 'Natural', 'Language', 'Processing’]

Using Tensorflow Tokenizer:

'hello': 1, 'class': 2, 'welcome': 3, 'to': 4, 'the': 5, 'world': 6, 'of': 7, 'natural': 8, 'language': 9, 'processing': 10

Src: **Natural Language Processing – Tokenization by LAURENCE MORONEY …**
https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%201.ipynb#scrollTo=zX4Kg8DUTKWO

Tokenization…

Tokenizers That Provide IDs: Integration with Machine Learning Models: Tokenizers that output tokens as IDs are often part of or designed to work closely with machine learning models, especially those in deep learning. Since these models don't process raw text but rather numerical data, each unique token is mapped to a unique ID. This ID then corresponds to either an index in an embedding matrix (where each ID can be used to look up the corresponding word embedding) or serves as a direct input to the model. Frameworks like TensorFlow, PyTorch, and libraries built on top of them (e.g., Hugging Face's Transformers) use this approach.

Tokenizers without IDs: General Text Processing: Some tokenizers are designed for more general text processing tasks, such as syntax parsing, POS tagging, or simply extracting textual features for non-machine-learning applications.

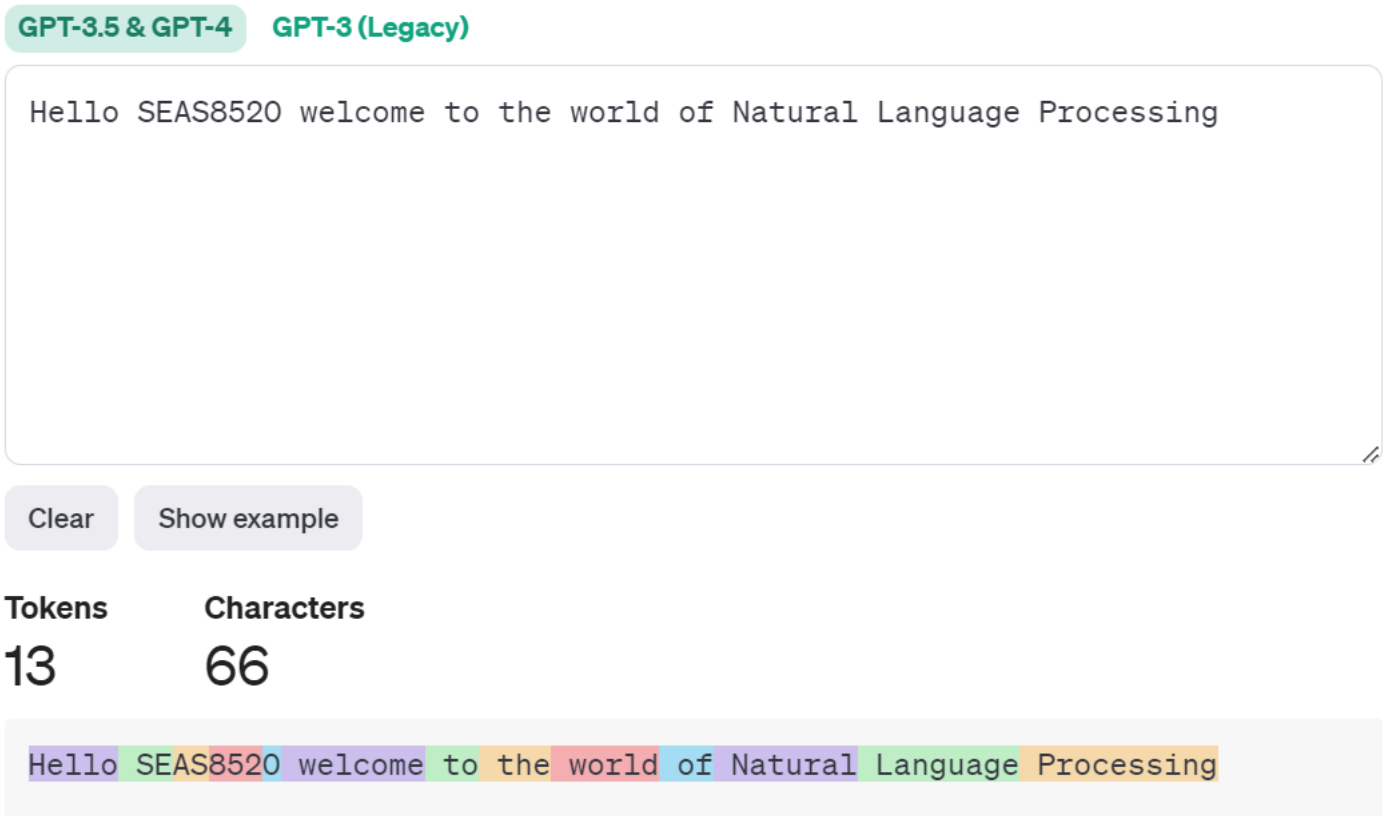
Src: **Natural Language Processing – Tokenization by LAURENCE MORONEY …**
https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%201.ipynb#scrollTo=zX4Kg8DUTKWO

Tokenization…

Let us see how ChatGPT does tokenization….

Src: **Natural Language Processing – Tokenization by LAURENCE MORONEY …**
https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%203%20-%20NLP/Course%203%20-%20Week%201%20-%20Lesson%201.ipynb#scrollTo=zX4Kg8DUTKWO

Tokenization…



A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly ¾ of a word (so 100 tokens ~= 75 words)

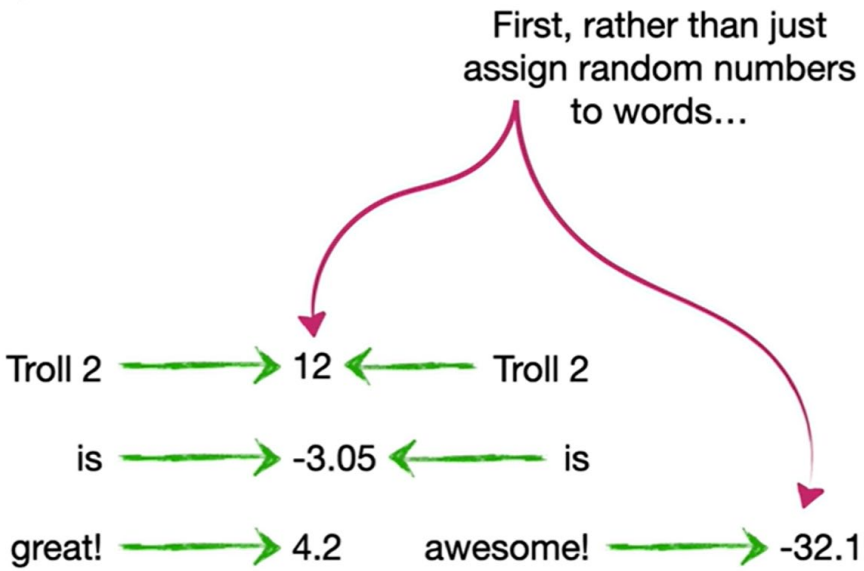
Embeddings…

- Tokenization enabled us to “chop down” the text into logical units (words)/Tokens.
- These IDs do not provide any “digital meaning” to the words.
- Hence the need for Embedding which is taking the words/tokens and mapping them to a higher dimensional space which carries meaning (more about that in a bit).
- How?

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Embeddings…

- Assume there are these two statements:
  - Troll 2 is great
  - Troll 2 is awesome
- Assume a single dimension embedding: may be assign random numbers??
- While Random numbers are "technically" OK, they do not enable any relationship discovery between great & awesome (as example), even though they are almost the same. So any ANN using these embeddings won't leverage one learning to the other.



Random Numbers Won't work

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

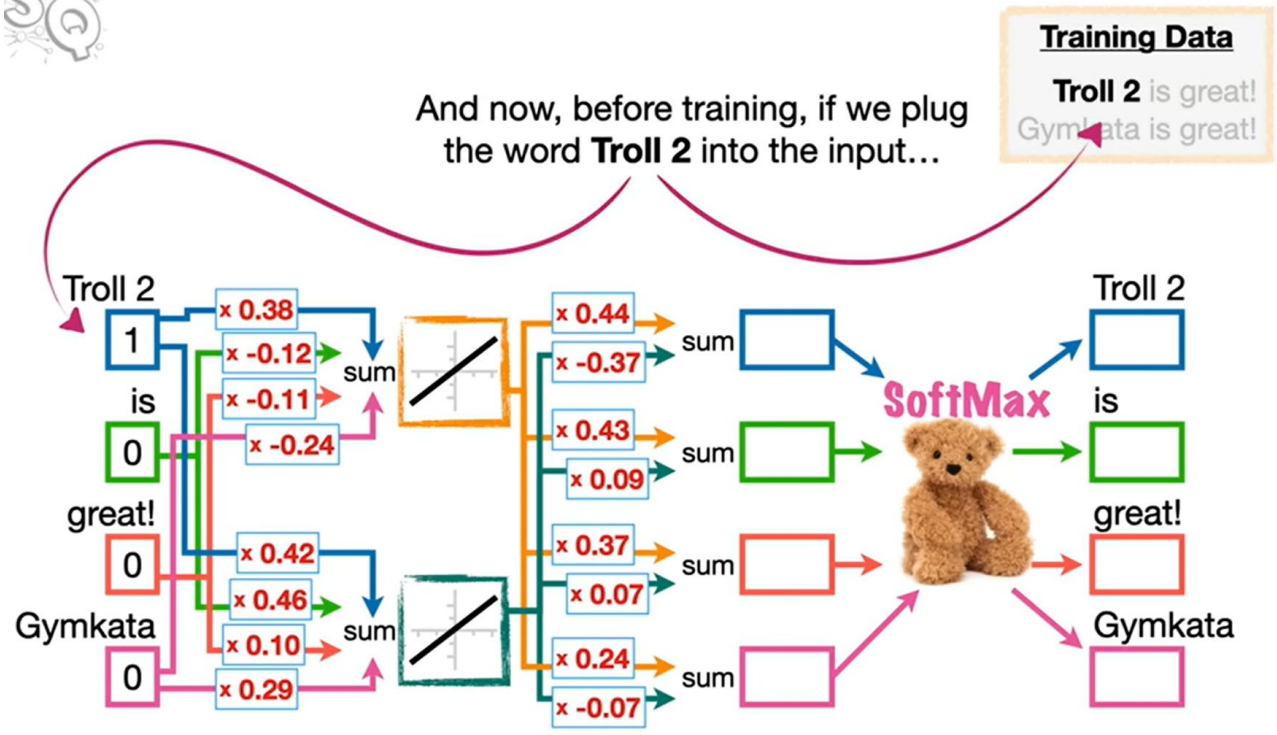
Embeddings…

- Since words have many different meanings depending on many factors especially the context on which they are used, practically, we need more than one dimension to keep track of all of these “contexts”.
- So how to come up with these different numerical values of the dimensions…
- Use an ANN: Given a specific word, it should predict the “next word”

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Embeddings…

Use an ANN: One hot-encoding. If a given “input” has 1 for the word, using the ANN and the trained weights, it should predict the next word.



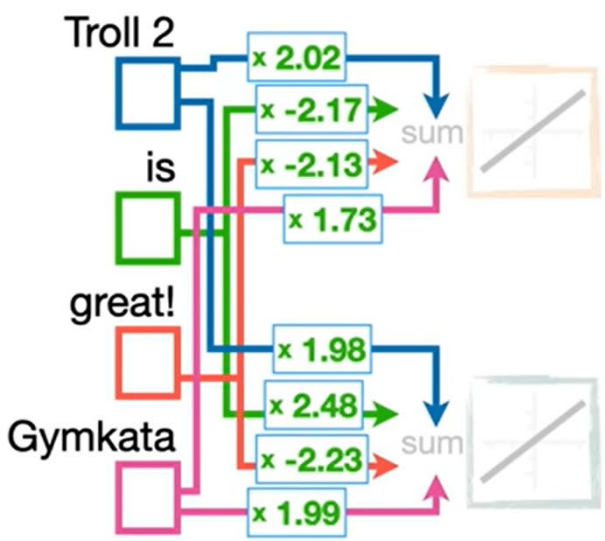
Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh *Strommer,* Statquestt: Word Embedding and Word2Vec clearly explained

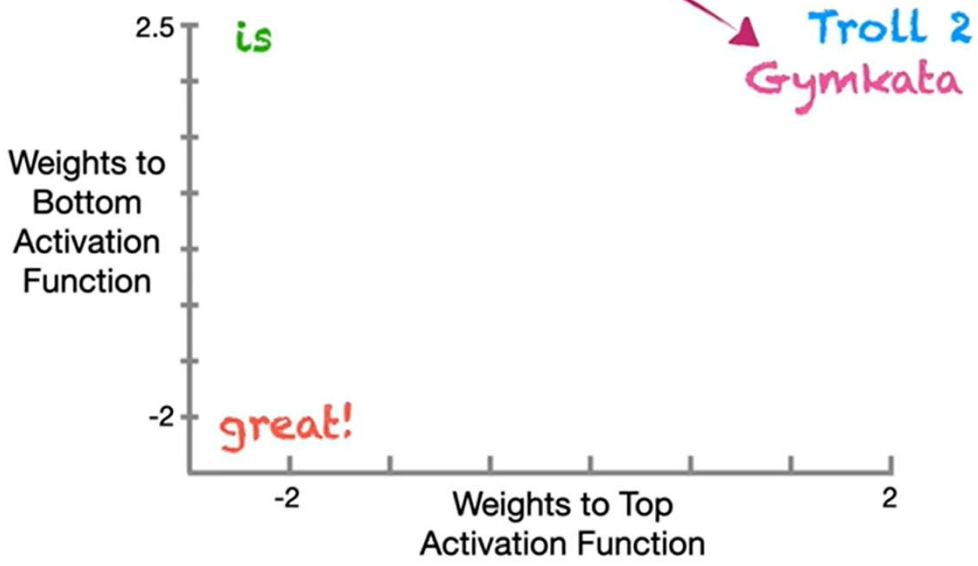
Embeddings…

…and this can result in similar words ending up with similar embeddings.

Training Data

Troll 2 is great!
Gymkata is great!



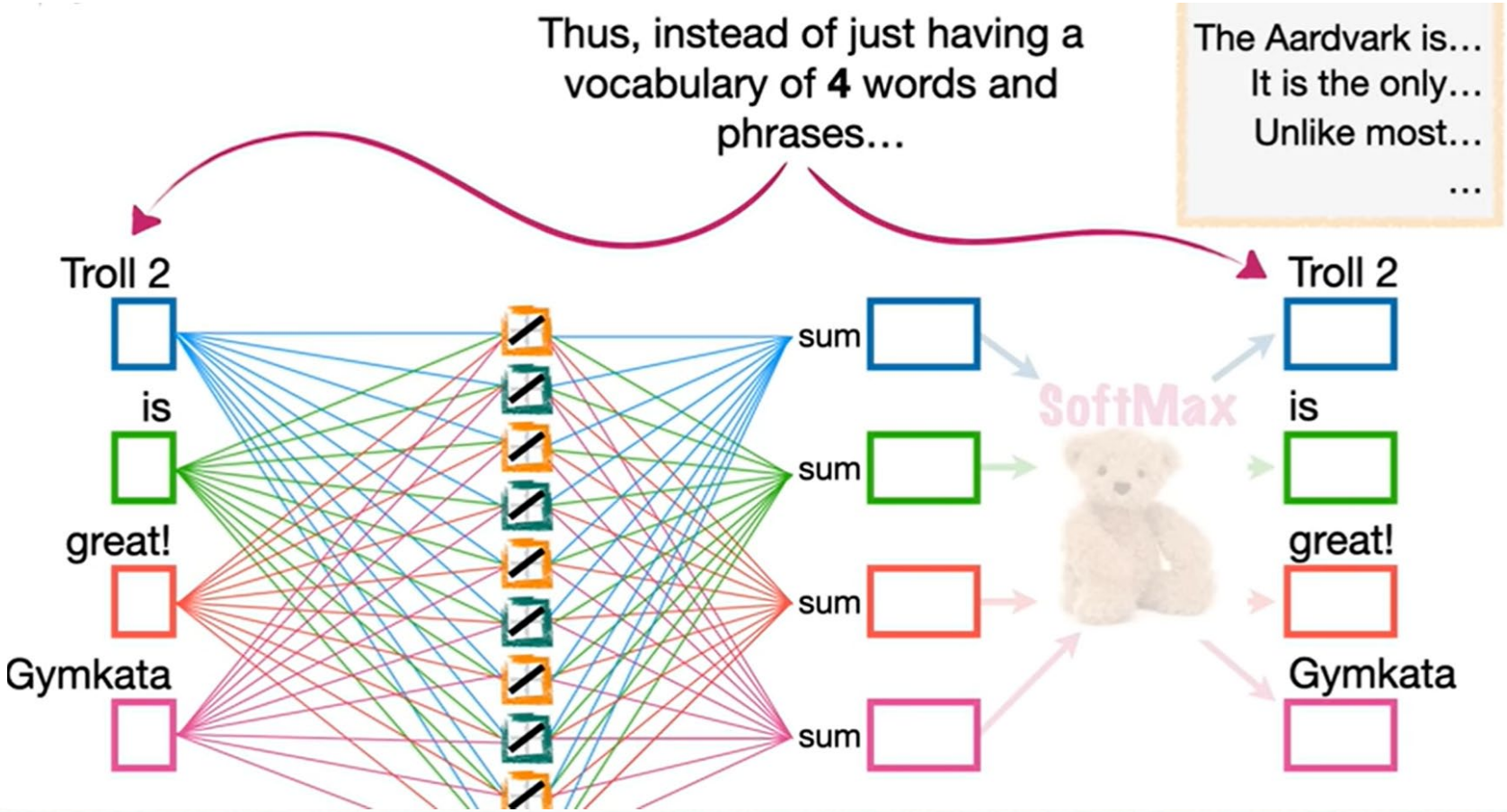


Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Embedding…

Training Data:

Wikipedia



Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Embedding…

- Embeddings convert tokens (typically words) into dense vectors of fixed size; i.e. It is like a digital “n vector size dictionary” of a word.
- Numerical Representation: Unlike the simple integer IDs from tokenization, word embeddings capture semantic information. Words with similar meanings will have embeddings that are close in the vector space.
- Embeddings are designed to capture semantic relationships between words in a dense, continuous vector space. This makes them highly useful for many NLP tasks where understanding the relationship between words is crucial.

Tokenization provides a structured format, while embeddings provide a semantically rich representation

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

https://projector.tensorflow.org/

Hands On…..

Word2Vec Embedding…

Continuous Bag of Words (CBOW): The CBOW model predicts a target word given its context. The "context" typically consists of a fixed-size window of surrounding words. For example, in the sentence "The cat sat on the mat," with a target word "sat" and a window size of 2, the context words would be ["The", "cat", "on", "the"].

The input layer consists of the context words, which are initially represented by one-hot encoded vectors. These vectors are projected onto a shared hidden layer but not combined; rather, they are averaged or summed, which leads to the "bag of words" notion, implying the order of words in the context does not affect the prediction. The model then uses the hidden layer to predict the target word.

The goal of the CBOW model is to accurately predict the target word based on the context words. During training, the model adjusts its weights to minimize the error in predicting the target word.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

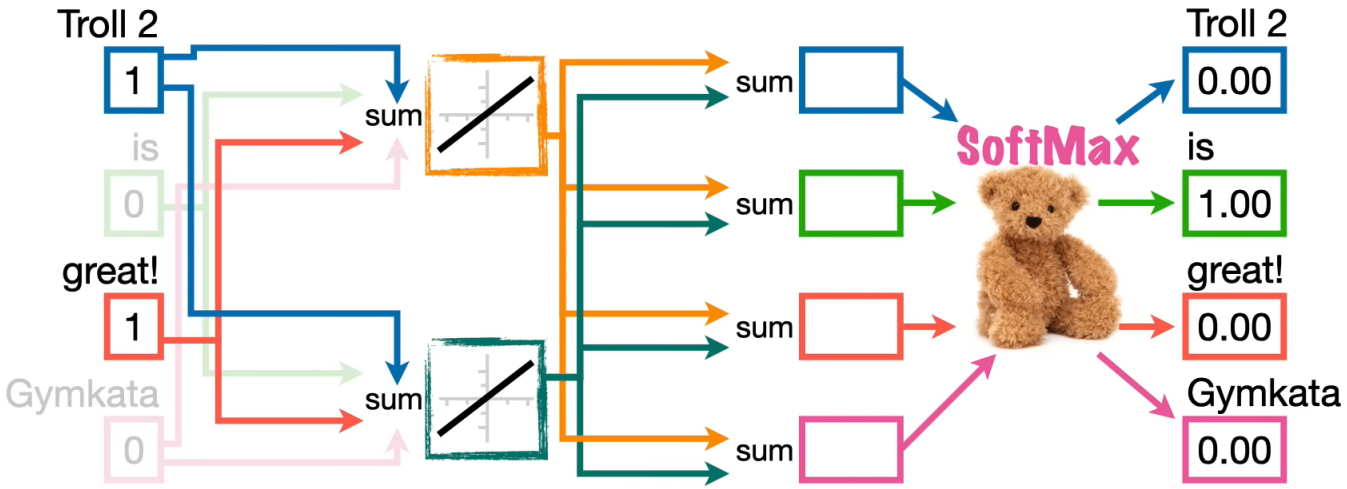
Word2Vec Embedding…

To Provide more context rather than predicting “next word”, word2vec uses continuous bag of words. i.e. it uses the surrounding words to predict the word in the middle.

The first method, called **Continuous Bag of Words**, increases the context by using the surrounding words to predict what occurs in the middle.

Training Data

Troll 2 is great!
Gymkata is great!



Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Word2Vec Embedding…

Skip-gram: here the model inverts the task of CBOW by predicting the surrounding context words for a given target word. For the same example sentence, with "sat" as the target word and the same window size, Skip-gram attempts to predict ["The", "cat", "on", "the"] based on "sat".

The input layer is the target word, represented by a one-hot encoded vector. This vector is projected onto a hidden layer, and from there, the model attempts to predict the context words in the output layer. Unlike CBOW, Skip-gram treats the prediction of each context word as a separate task, effectively decomposing the problem into multiple prediction tasks, one for each context word.

The aim of the Skip-gram model is to produce embeddings that are good at predicting the surrounding words of each target word in the corpus. It adjusts its weights to minimize the error in these predictions.

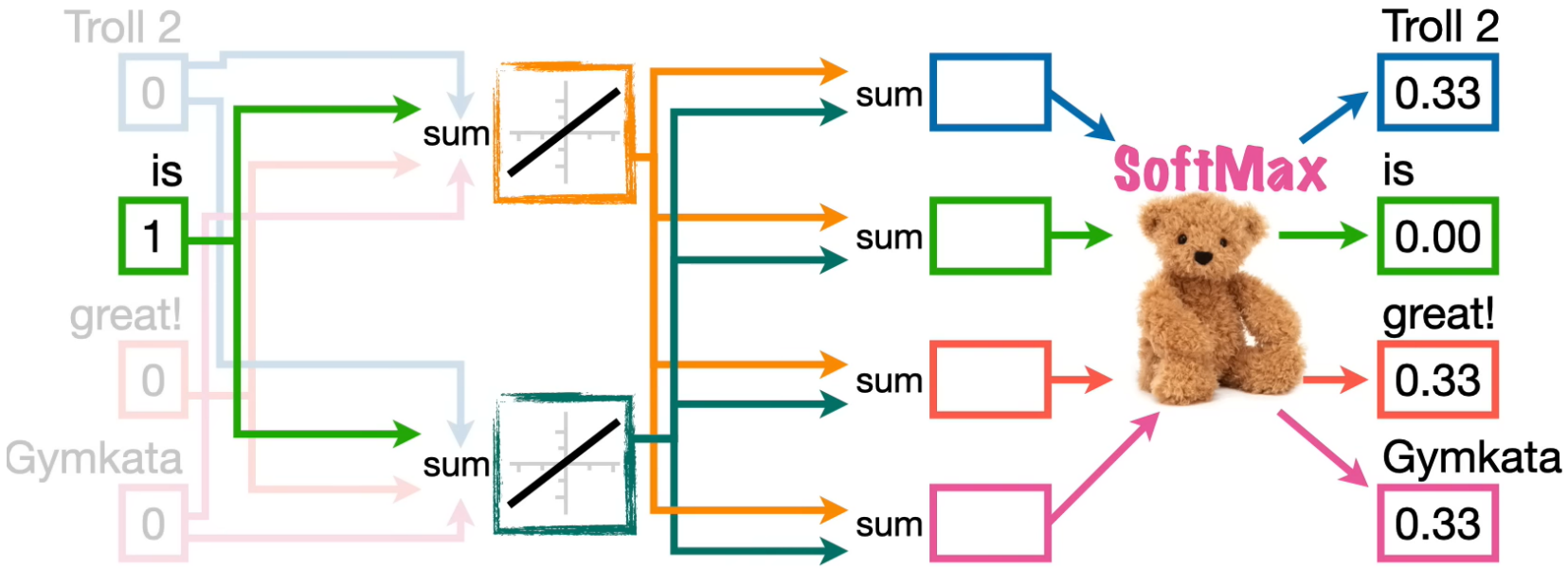
Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Word2Vec Embedding…

The second method, called **Skip Gram**, increases the context by using the word in the middle to predict the surrounding words.

Training Data

Troll 2 is great!
Gymkata is great!



Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Word2Vec Embedding…

Skip-gram: The number of words predicted around the given word is determined by the size of the "window" or "context window." This window size is configurable and is a key hyperparameter when training a Skip-gram model.

The window size specifies the maximum distance between the target word and a context word within a sentence that will be considered as positive pairs during training. For example, if the window size is set to 2, then for the target word, the model tries to predict up to 2 words before and 2 words after the target word in the sentence.

Example: Consider the sentence "The quick brown fox jumps over the lazy dog", and suppose we're focusing on the target word "fox" with a window size of 2. The positive training pairs for "fox" would be: (fox, quick) (fox, brown) (fox, jumps) (fox, over)

Embeddings…

- CBOW is faster and somewhat more efficient than Skip-gram because it reduces the context words into a single prediction task. However, Skip-gram tends to perform better with smaller datasets and is more effective at capturing infrequent words.

In Current Product Embeddings:

- Of course, the network will have more than 2 activation functions
- And the corpus will be more than a couple of sentences (English encyclopedia, Wikipedia….etc)
- Many Embeddings are available to download
- Some NLP models have their own embeddings which are accessible thru APIs.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

The Law of Similarity in NLP

•The law of similarity is a principle stating that similar items are perceived as a group.

•In language processing, this translates to grouping similar words or phrases.

•Similarity affects how language models understand and generate language.

•Measures of similarity in language include semantic and contextual closeness.

•ChatGPT and other language models leverage this law to improve coherence and relevance in conversation.

Dot Product - The First Step to Similarity

•The dot product is a mathematical operation that takes two equal-length sequences of vectors and returns a single number.

•Algebraically, it sums up the products of corresponding elements of vectors.

•Geometrically, it measures the magnitude of the projection of one vector onto another.

•The dot product increases as the angle between two vectors decreases, which indicates more similarity.

•In NLP, a higher dot product between word vectors suggests greater semantic similarity.

Dot Product - The First Step to Similarity

Algebraic definition:

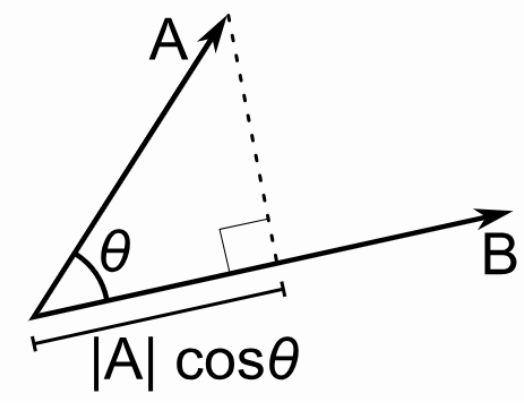
For two vectors, the dot product is the sum of the products of corresponding components of them.

Think of two vectors a and b as below:

$a=[a_1 a_2 ... a_n]$ $b=[_b 1 b_2 ... b_n]$

Then, the dot product of a and b becomes:

$a \cdot b = \sum a_i b_i$ for I = 1 to n = $a_1 b_1 + a_2 b_2 + a_3 b_3$



Projection of A onto B

Geometric definition

$a \cdot b = |a||b| \cos\theta$

Geometrically, the dot product is the product of the magnitudes of two vectors and the cosine of the angle between two.

Cosine Similarity - Normalizing the Dot Product

•Cosine similarity is derived from the dot product normalized by the magnitudes of the vectors.

•It calculates the cosine of the angle between two vectors, providing a measure of orientation similarity.

•This metric is not influenced by the magnitude or length of the vectors, focusing solely on direction.

•A cosine similarity of 1 means words are semantically identical, while 0 means no similarity.

•LLMs uses cosine similarity to find the best-matching words and phrases during conversation.

Cosine Similarity - Normalizing the Dot Product

•Understanding the degree of similarity between words is crucial for language models like ChatGPT.

•These measures help in tasks such as word prediction, sentence completion, and text summarization.

•They improve the model's ability to capture nuances in language and provide contextually appropriate responses.

•Cosine similarity is used in attention mechanisms to weigh the influence of different words.

Measuring Similarity

There are other methods for similarity measurements e.g. Manhattan Distance, Euclidean Distance..

Word2Vec Embedding…

```
# Now, let's consider two different contexts
sentence1 = "I sat by the river bank."
sentence2 = "I deposited money in the bank."
# As Word2Vec doesn't consider context, the embeddings for 'bank' in both sentences
will be the same
embedding_from_sentence1 = get_word_embedding('bank')
embedding_from_sentence2 = get_word_embedding('bank')
# Calculate cosine similarity to demonstrate they are identical
similarity = 1 - cosine(embedding_from_sentence1, embedding_from_sentence2)
```

It is 1 i.e. they are the same i.e. there is no context

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Word2Vec Embedding…

So Bank Embedding is an array of 300 elements in length and floating point

- <class 'numpy.ndarray'> 300
- Float32
- Let us take a look…

Word2VecBankEmbedding.txt

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Word2Vec Embedding…

Word2Vec employs two main algorithms to create embeddings, but not simultaneously. You choose one of the two when training your model:

Continuous Bag of Words (CBOW): In this method, the model predicts the current word from a window of surrounding context words. The order of context words doesn't influence prediction.

CBOW takes the average of the context word embeddings and predicts the target word.

CBOW is faster and works well with smaller datasets.

Skip-Gram:

Here, the model predicts surrounding context words from the current word. In essence, it's the opposite of CBOW.

Skip-Gram uses the embedding of the target word to predict context words.

Skip-Gram works well with larger datasets and tends to capture more intricate word representations, especially with infrequent words

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Embeddings…

GloVe (2014): The Global Vectors for Word Representation (GloVe) method was introduced by researchers at Stanford. Unlike Word2Vec, which is predictive, GloVe is a count-based method that factorizes the word-context co-occurrence matrix.

FastText (2016): Developed by Facebook's AI Research (FAIR) lab, FastText is an extension of Word2Vec. It represents words as bags of character n-grams, allowing it to generate better embeddings for morphologically rich languages and out-of-vocabulary words.

ELMo (2018): Embedded from Language Models (ELMo) embeddings, developed by AllenAI, are deep contextualized word representations. Unlike Word2Vec and GloVe, which assign the same vector to a word regardless of its context, ELMo generates different vectors for a word based on its usage in a sentence. This made ELMo embeddings particularly powerful for many NLP tasks.

Transformer (2017): The "Attention Is All You Need" paper by Vaswani et al. introduced the Transformer architecture. This was a departure from the recurrent layers (like LSTM) commonly used in NLP tasks.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

GloVe Embedding…

Let us check what Stanford has for GloVe.

https://nlp.stanford.edu/projects/glove/

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Hands On…..

Embeddings…

Step 1: Define a Small Corpus:

- "the cat sat on the mat"
- "the dog sat on the log"
- "cats and dogs are friends"

Step 2: Preprocess the Data

- Tokenization: Convert each sentence into a sequence of tokens (or words).
- Build Vocabulary: Create a mapping from words to unique indices.
- Sequence Creation: For each sentence, create multiple input-target pairs. For example, from "the cat sat on the mat", create pairs like ("the", "cat"), ("the cat", "sat"), etc.

Step 3: Create Training Examples

- Use the sequences to create input and target pairs, where each input is a sequence of word indices and the target is the index of the next word.

Embeddings…

Step 4: Build the Model

- Embedding Layer: Maps the indices of words to their learned embeddings.
- Dense Layer: A fully connected layer that predicts the next word from the embeddings.

Step 5: Train the Model

- Train the model on the input-target pairs. This will adjust the embeddings to capture semantic relationships between words.

GPT Embeddings… APIs

Interacting with ChatGPT to obtain embeddings allows users to leverage the full scale of the model's capabilities, providing rich, context-aware representations of text. These embeddings encapsulate the model's extensive knowledge base and understanding of language nuances, offering valuable insights for a wide range of NLP tasks.

BERT

- BERT: Bidirectional Encoder Representations from Transformers
- Training Data: BooksCorpus (800 million words) and English Wikipedia (2,500 million words)
- Model Sizes: BERT-Base: 12 layers (transformer blocks) 110 million parameters

  BERT-Large: 24 layers (transformer blocks) 340 million parameters
- **Key Features**:
  - Bidirectional context understanding
  - Pre-trained on masked language modeling
  - Fine-tuned for various tasks (classification, QA, etc.)
- **Innovations**: Introduced "Attention Masking" for dynamic input lengths
- Captures deep contextual relations between words

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: https://statquest.org/transformer-neural-networks-chatgpts-foundation-clearly-explained/

BERT

- BERT utilizes the Transformer architecture and was originally trained on vast amounts of text, including the entire English Wikipedia and BooksCorpus.
- While pre-trained on general data, BERT can be fine-tuned on a specific dataset to capture domain-specific nuances and context. This adaptability has made it a powerful tool for various NLP tasks.
- The Hugging Face Model Hub hosts thousands of BERT variants and models fine-tuned for diverse tasks and languages.
- https://huggingface.co/models?sort=trending

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: https://statquest.org/transformer-neural-networks-chatgpts-foundation-clearly-explained/

BERT Embedding…

Using BERT , the embedding for Bank differs between the two sentences…

sentence1 = "I sat by the river bank."

sentence2 = "I deposited money in the bank."

embedding1 = get_word_embedding(sentence1, "bank")

embedding2 = get_word_embedding(sentence2, "bank")


```
similarity = torch.nn.functional.cosine_similarity(
    torch.tensor(embedding1).unsqueeze(0), torch.tensor(embedding2).unsqueeze(0)
)
```

Cosine similarity between the embeddings: 0.525728702545166

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: https://statquest.org/transformer-neural-networks-chatgpts-foundation-clearly-explained/

BERT - Masked Language Model

- MLM enables/enforces bidirectional learning from text by masking (hiding) a word in a sentence and forcing BERT to bidirectionally use the words on either side of the covered word to predict the masked word.
- **Examples:**
- Dang! I'm out fishing and a huge trout just [blank] my line!
- What could a "good" prediction for that interruption … Broke.
- A random 15% of tokenized words are hidden during training and BERT's job is to correctly predict the hidden words. Thus, directly teaching the model about the English language.
- 3.3 Billion words has contributed to BERT's continued success

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

https://towardsdatascience.com/a-gentle-introduction-to-natural-language-processing-e716ed3c0863

BERT – Next Sentence Validity/Prediction

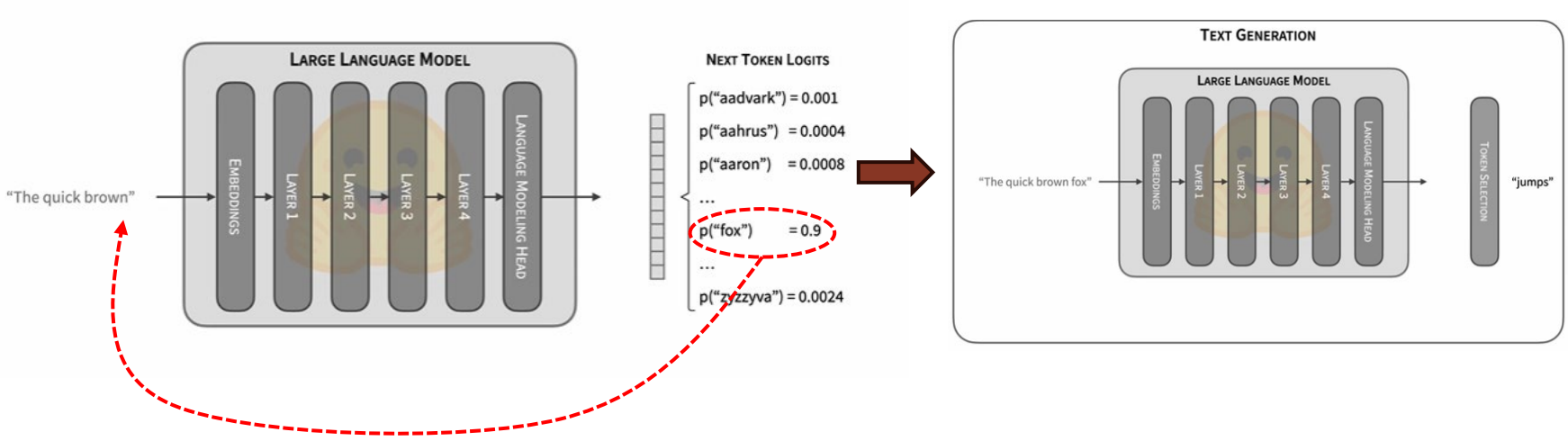
- NSP/NSV (Next Sentence Prediction) is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not.
- **Examples:**
- Paul went shopping. He bought a new shirt. (correct sentence pair)
- Ramona made coffee. Vanilla ice cream cones for sale. (incorrect sentence pair).
- In training, 50% correct sentence pairs are mixed in with 50% random sentence pairs to help BERT increase next sentence prediction accuracy.
- BERT is trained on both MLM (50%) and NSP (50%) at the same time

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

https://huggingface.co/blog/bert-101#2-how-does-bert-work

LM– Next Word Prediction

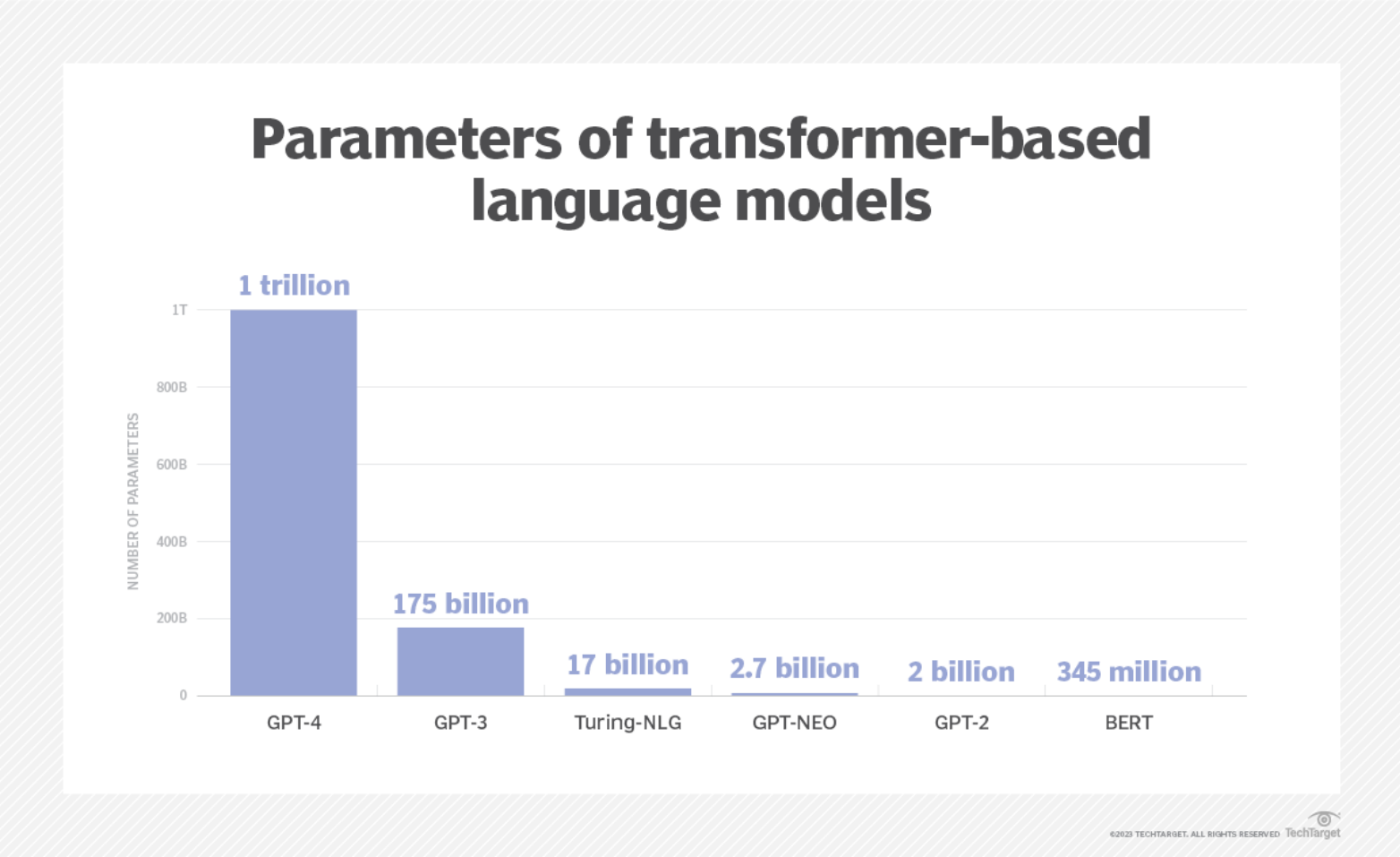
- Language models take a sequence of text tokens as input and returns the probability distribution for the next token.



- The process depicted above is repeated iteratively until some stopping condition is reached (End of Sequence Token, or Max Length)

https://huggingface.co/docs/transformers/llm_tutorial

LLMs… Billion Plus Params



Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
https://www.techtarget.com/whatis/definition/large-language-model-LLM

What are LLMs

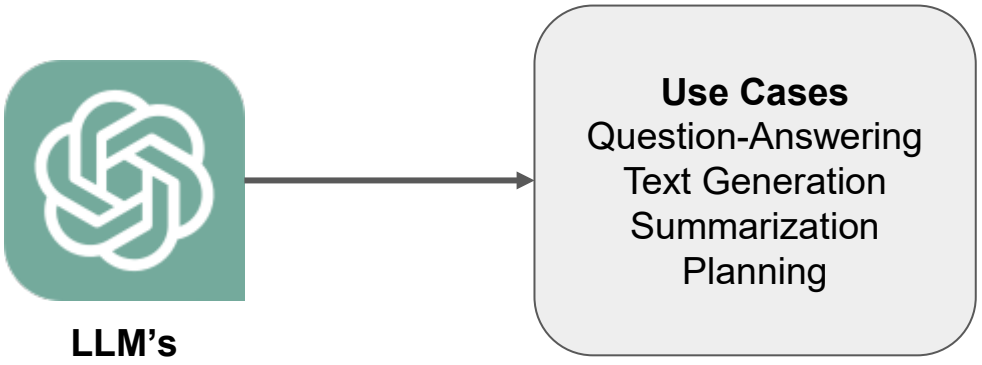
A large language model (LLM) is a type of artificial intelligence (AI) algorithm that uses deep learning techniques and massively large data sets to understand, summarize, generate and predict new content. The term generative AI also is closely connected with LLMs, which are, in fact, a type of generative AI that has been specifically architected to help generate text-based content.

An LLM typically has at least one billion or more parameters

https://www.techtarget.com/whatis/definition/large-language-model-LLM

What are LLMs

LLMs are a phenomenal piece of technology for knowledge generation and reasoning. They are pre-trained on large amounts of **publicly available data**.



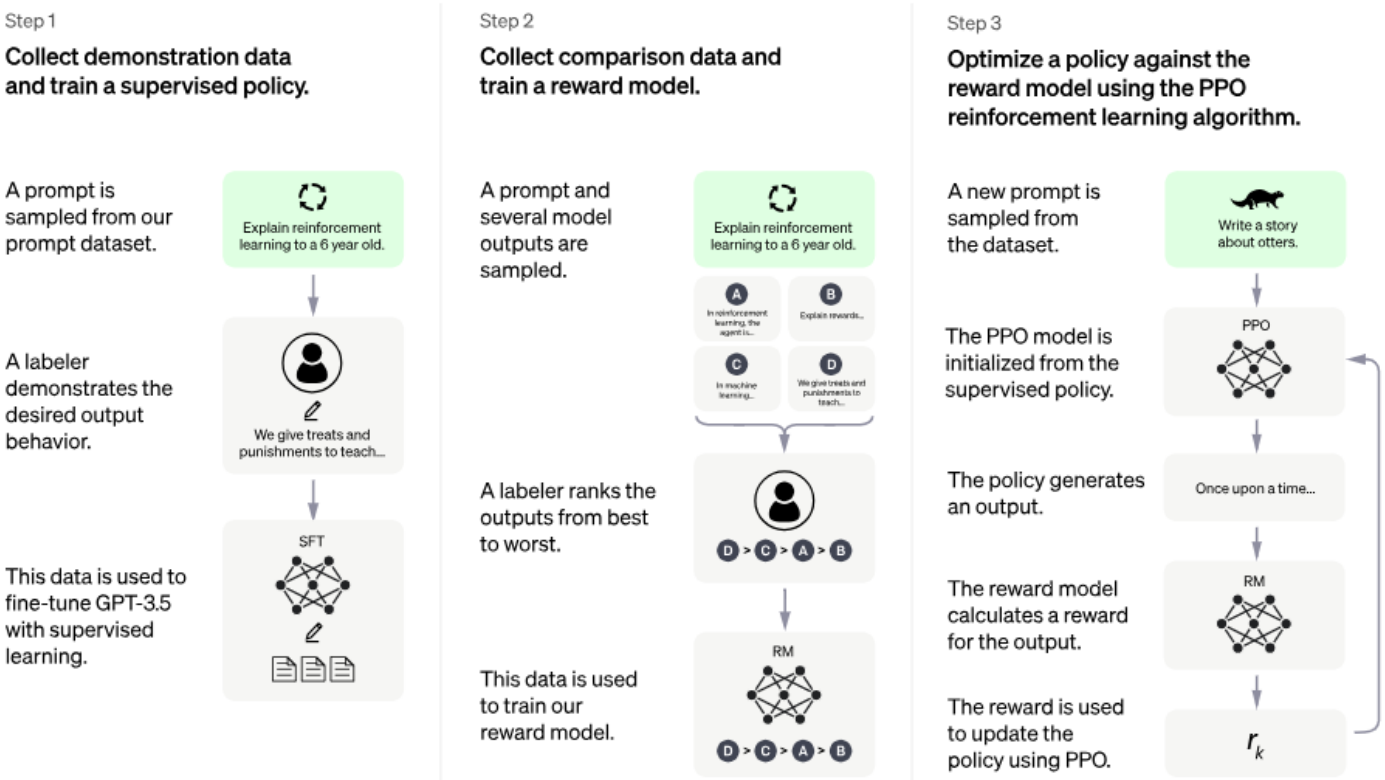
Adopted from: **Using LlamaIndex for building Generative AI apps**
https://www.youtube.com/watch?v=YN6n5hvmsx8

LLMs in the security space

- By 2025, two-thirds of businesses will leverage a combination of generative AI and RAG to power domain-specific, self-service knowledge discovery, improving decision efficacy by 50% (NVIDIA)
- organizations are challenged in training new and existing employees. With copilots, cybersecurity professionals can get near real-time responses and guidance on complex deployment scenarios without the need for additional training or research
- chatbots that can retrieve the most up-to-date information in real-time and provide accurate responses in natural language
- Using generative AI, it's possible to improve vulnerability defense while decreasing the load on security teams

https://developer.nvidia.com/blog/bolstering-cybersecurity-how-large-language-models-and-generative-ai-are-transforming-digital-security/

GPT 3.4



The model was trained using Reinforcement Learning from Human Feedback (RLHF)

https://openai.com/blog/chatgpt

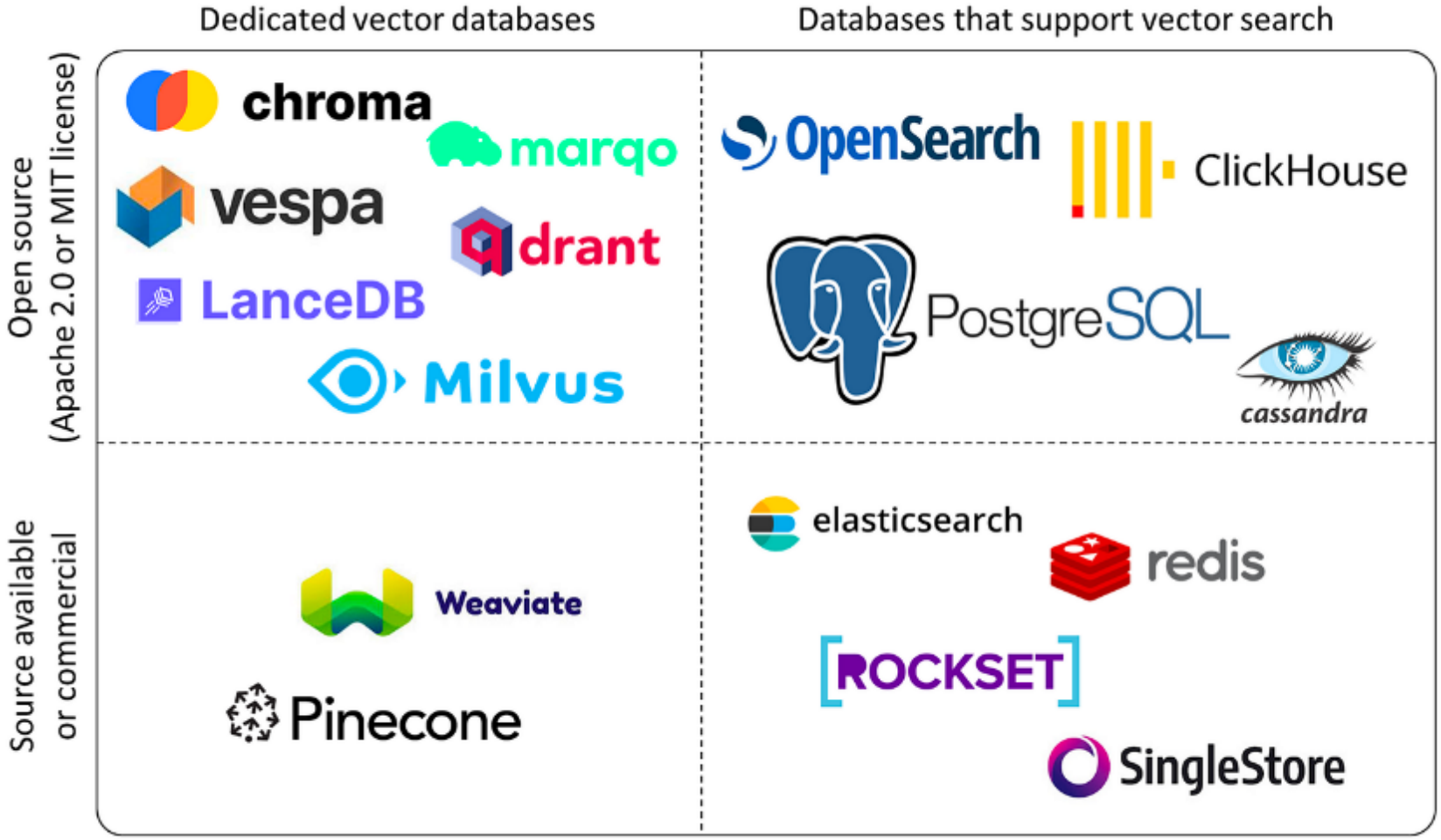
Some areas for LLMs

While the ideas to leverage LLMs are limitless, some key areas include:

- Content Generation (including Graphics)
- Summarization
- Translations
- Code Generation
- Gaming….
- Etc…
- Idea Generation: A great opportunity for coming up with suggestions.

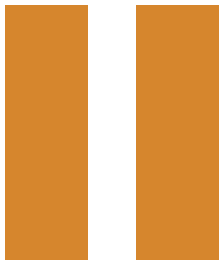
https://openai.com/blog/chatgpt

Vector DBs



Src: https://www.datacamp.com/blog/the-top-5-vector-databases

Back-up



BREAK

Please come back @

10:15 AM EST

9:15 AM CST

References

In addition to the references in each slide the below are leveraged throughout this course.

Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

VanderPlas, J. (2017). Python Data Science Handbook. O'Reilly.
Wolff, S. G. (2018). Less is more: optimizing classification performance through feature selection in a very-high-resolution remote sensing object-based urban application. GIScience & Remote Sensing. doi:10.1080/15481603.2017.1408892

Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.
Josh Stormer, https://statquest.org/
Chollet, F. (2021). Deep Learning with Python, Second Edition (2nd ed.). Manning