Large Language Models

- Web-Search - Recap
- Statistical Language models
- N-Gram Language Models & Translation Models
- Word Embeddings
- Contextual Word Embeddings

Development:
Moshe Friedman

Credits:
Yoav Goldberg, Ido Dagan, Reut Tsarfaty , Moshe Koppel, Wei Song, David Bamman, Ed Grefenstette, Chris Manning, Tsvi Kuflik, Hinrich Schütze, Christina Lioma and more

Web Search - Recap

The Paradigm Shift

Classic IR (Library Science)

Context: A lawyer searching Case Law.

Problem: Scarcity. There might be only one relevant precedent.

Goal: Find everything. Missing a document is expensive.

Metric: High Recall.

Web Search

Context: "Chocolate Cake Recipe".

Problem: Abundance. There are 5 million relevant recipes.

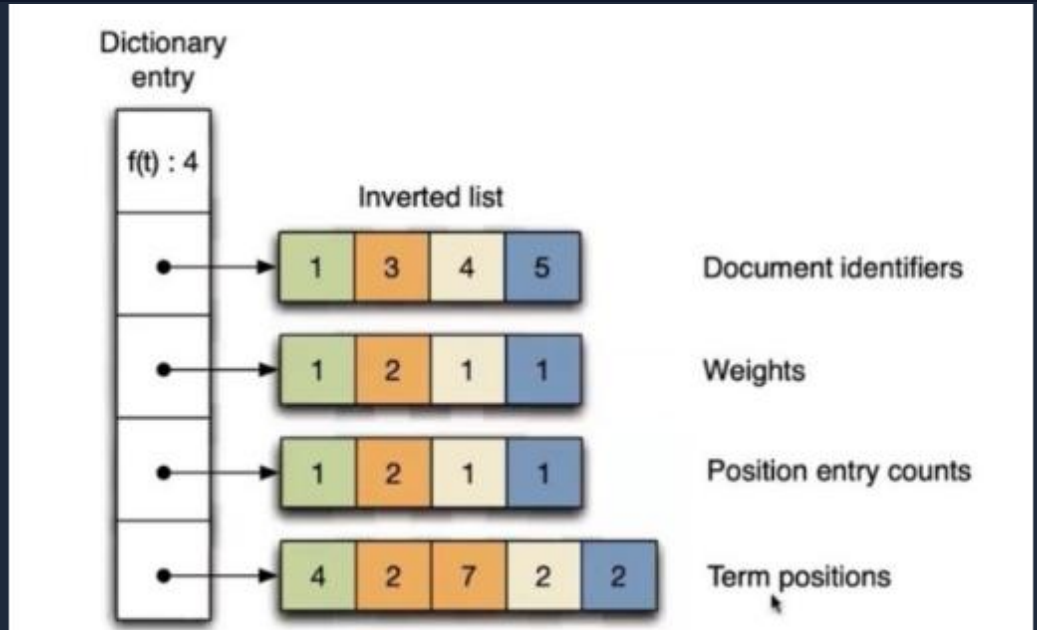
Goal: Find the *best* one immediately.

Metric: High Precision (specifically at the top).

Classic IR Pipeline

The standard process for static document collections (e.g., legal corpuses).

1. **Tokenization:** Splitting text into words.
2. **Normalization:** Lowercasing, removing punctuation.
3. **Stopword Removal:** Removing "the", "and“ & "is".
4. **Stemming/Lemmatization:** "Running" -> "Run".
5. **Indexing:** Creating the Inverted Index.



B. Cambazoglu and R. Baeza-Yates, "Scalability Challenges in Web Search Engines"

The Spider's Loop

1. Pop URL from Frontier.

2. Check Robots.txt & Politeness.

3. Fetch HTML (DNS resolution + Download).

4. Parse content & extract links.

5. Dedup (Have we seen this link?).

6. Push new links to Frontier.

7. Index the text.

Anatomy of a URL

Uniform Resource Locators (URLs) are the addresses of the web. Understanding their structure is vital for crawling.

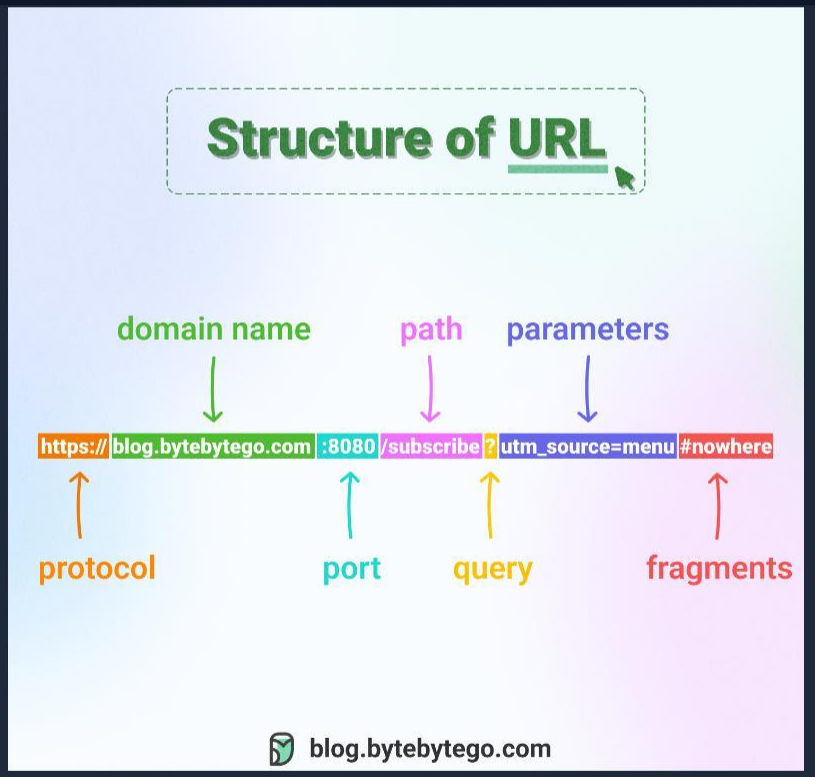
`https://` **Protocol:** How to transfer (HTTP/S).

`www.example.com` **Host/Domain:** The server address.

`/path/to/page` **Path:** Resource location on server.

`?id=123` **Query:** Parameters for dynamic pages.

`#section` **Fragment:** Anchor within the page (not sent to server).



Politeness & Robots.txt

DoS vs DDoS

DoS: Single crawler hammering a server. Crash caused by speed.

DDoS: Distributed crawler (1000 machines) hitting a server at once.

Solution: Crawl-delay and Robots.txt.

```
User-agent: * Disallow: /admin/ Crawl-delay: 5
User-agent: BadBot Disallow: /
```

Crawling Strategy: BFS

Breadth-First Search

Goal: General Coverage.

Logic: Use a FIFO Queue. Crawl level *d* before level *d+1*.

Why? Prevents getting stuck in "rabbit holes" (infinite calendars or archives) on a single site.

Example

- Seed: Home (d=0)
- Q: [Sports, News] (d=1)
- Visit Sports: Adds [Article A, Article B]
- Q: [News, Article A, Article B]
- Visit News: (Crucial: We visit News *before* Article A).

The "Seed" Problem

You can't crawl the web if you don't know where to start. The web graph must be traversed from known entry points.

What makes a good seed?

- **High Out-Degree:** Links to many other pages.
- **Trustworthy:** Not spam.
- **Centrality:** Stable and long-standing.
- **Topical Coverage:** Represents the domain well.

Strategies for Finding Seeds

Directories

DMOZ (archived), BOTW, and niche directories are human-curated lists of quality sites.

Institutions

University homepages (.edu) and Government portals (.gov) are high-trust hubs.

Inverse Links

Use existing indices (like Ahrefs/Majestic) to find who links *to* known authorities.

#Social

Highly shared URLs on Twitter/Reddit often point to fresh, relevant content.

Curated Lists

"Best of" blog posts and Wikipedia "External Links" sections.

HITS

Run HITS on a small set to identify "Hubs", then use those Hubs as seeds for a larger crawl.

The Language of the Web

HTML (Structure)

The Nouns & Verbs.

```
Title Link
```

This is what crawlers care about most.

CSS (Presentation)

The Adjectives.

```
.price { color: green; }
```

Crawlers use this to detect hidden text (spam).

JS (Behavior)

The Actions.

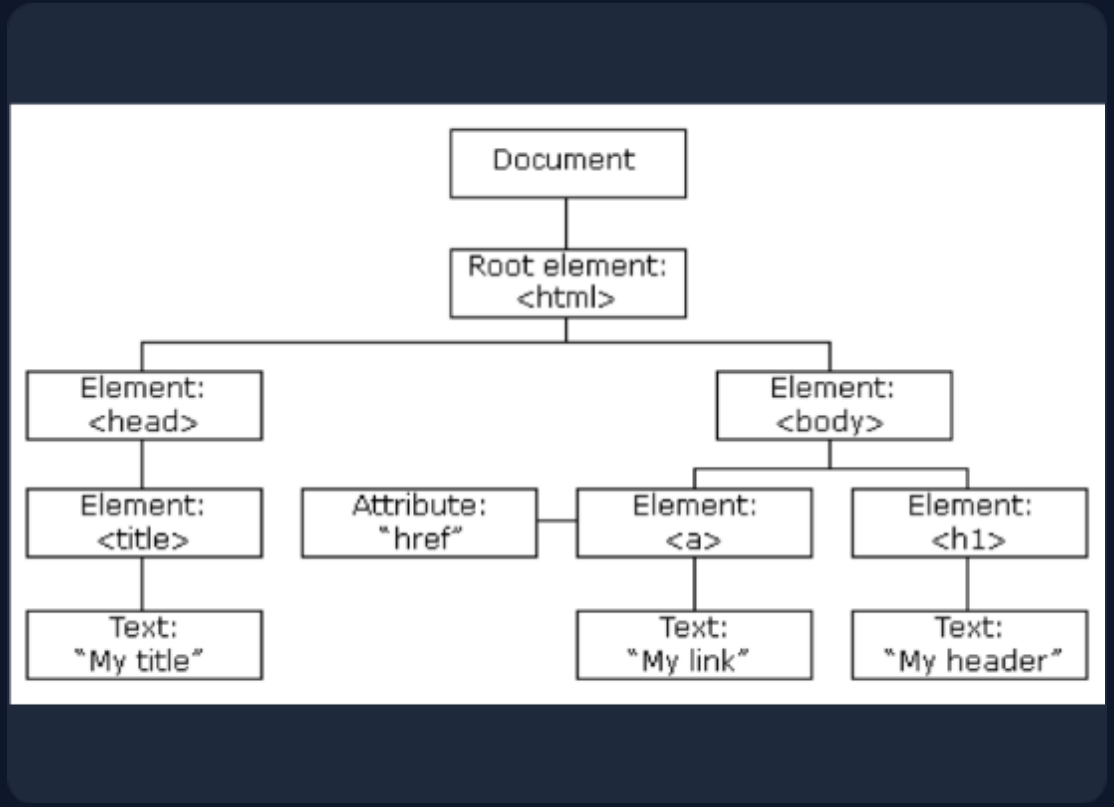
```
btn.onclick = () ⇒ {
fetch('/data') }
```

Challenge: Crawlers must execute JS to see content on modern sites.

HTML: The Language of Search

Search engines don't "see" pages like humans. They parse the **DOM** (Document Object Model).

- **Tags:** , provide semantic weight.
- **Links:** are the edges of the web graph.
- **Meta:** provides snippets.



Scraping vs. Crawling

Crawling (Discovery)

Goal: Discovery & Indexing.

Scope: Broad / The whole web.

Output: A list of URLs and cached text.

Example: Googlebot, Bingbot.

Tool: requests, Scrapy.

Scraping (Extraction)

Goal: Extraction.

Scope: Narrow / Specific sites.

Output: Structured data (prices, names).

Example: Price comparison, Lead gen.

Tool: BeautifulSoup (Static), Selenium (Dynamic).

Static Scraping

The Mechanics

Pros: Very fast, low CPU usage.

Cons: Cannot see content loaded by JavaScript.

Libraries: requests, BeautifulSoup.

```
import requests from bs4
import BeautifulSoup

# 1. Fetch
raw HTML response =
requests.get("https://example.com")

# 2. Parse
soup = BeautifulSoup(response.text,
'html.parser’)
```

```
# 3. Extract specific element  (e.g. $19.99
price)
price = soup.find('span', class_='price').text
print(price) # Output: $19.99
```

Dynamic Scraping

The Mechanics

Pros: WYSIWYG. Can handle infinite scroll and SPAs (React/Vue).

Cons: Slow. Launches a full browser instance.

Libraries: Selenium, Puppeteer.

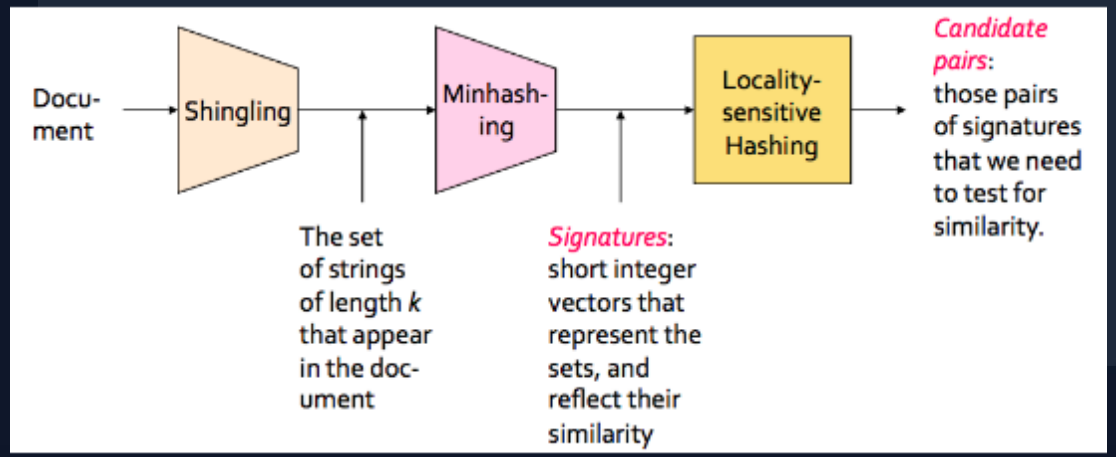
```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

# 1. Launch Browser
driver = webdriver.Chrome()
driver.get("https://example.com/dynamic")

# 2. Wait for JS to render
time.sleep(2)

# 3. Extract content generated by JS
price = driver.find_element(By.CLASS_NAME, "price").text
print(price) driver.quit()
```

Near-Duplicate Detection

The Problem

Approximately 30% of the web consists of near-duplicates (mirrors, revisions, spam). Exact match detection fails on dynamic timestamps or minor edits.

Exact hash matching (MD5) fails if one-byte changes (e.g., dynamic timestamp).



The Solution: Shingling

Shingles: Convert text into sets of N-grams (e.g., "a rose is a rose").

Jaccard Coefficient: Measure similarity by dividing the intersection of shingle sets by their union.

MinHashing: A technique to approximate Jaccard similarity efficiently at scale.

PageRank (PR): The Random Surfer

Imagine a surfer clicking random links forever.

PageRank (PR) is the probability that the surfer is on a specific page at any given time.

More incoming links = Higher probability.

Links from high-probability pages = More weight.

Damping factor (d): Probability of clicking (usually around 0.85).

Teleportation (1-d): Prevents getting stuck in dead ends (Sink nodes).

HITS: Hubs & Authorities

Proposed by Kleinberg. Query-dependent.

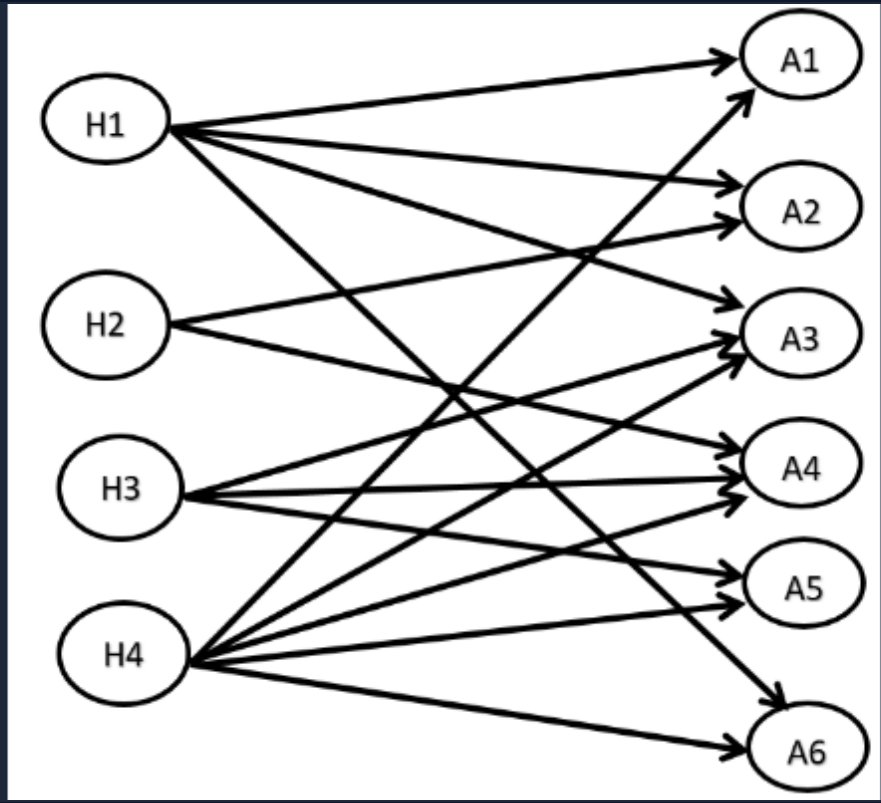
Authorities: Pages with good content (linked to by many hubs).

Hubs: Pages with good lists of links (link to many authorities).

Mutually reinforcing:

Good hubs point to good authorities.

Good authorities are pointed by good hubs.



The Spam Arms Race

If ranking = money, people will cheat.

Content Spam

Keyword stuffing, hidden text, scraped content, auto-generated gibberish.

Link Spam

Link farms, paid links, comment spam, buying expired domains.

Cloaking

Showing search engines one version of a page and users another (e.g., porn/gambling).

Machine Learning

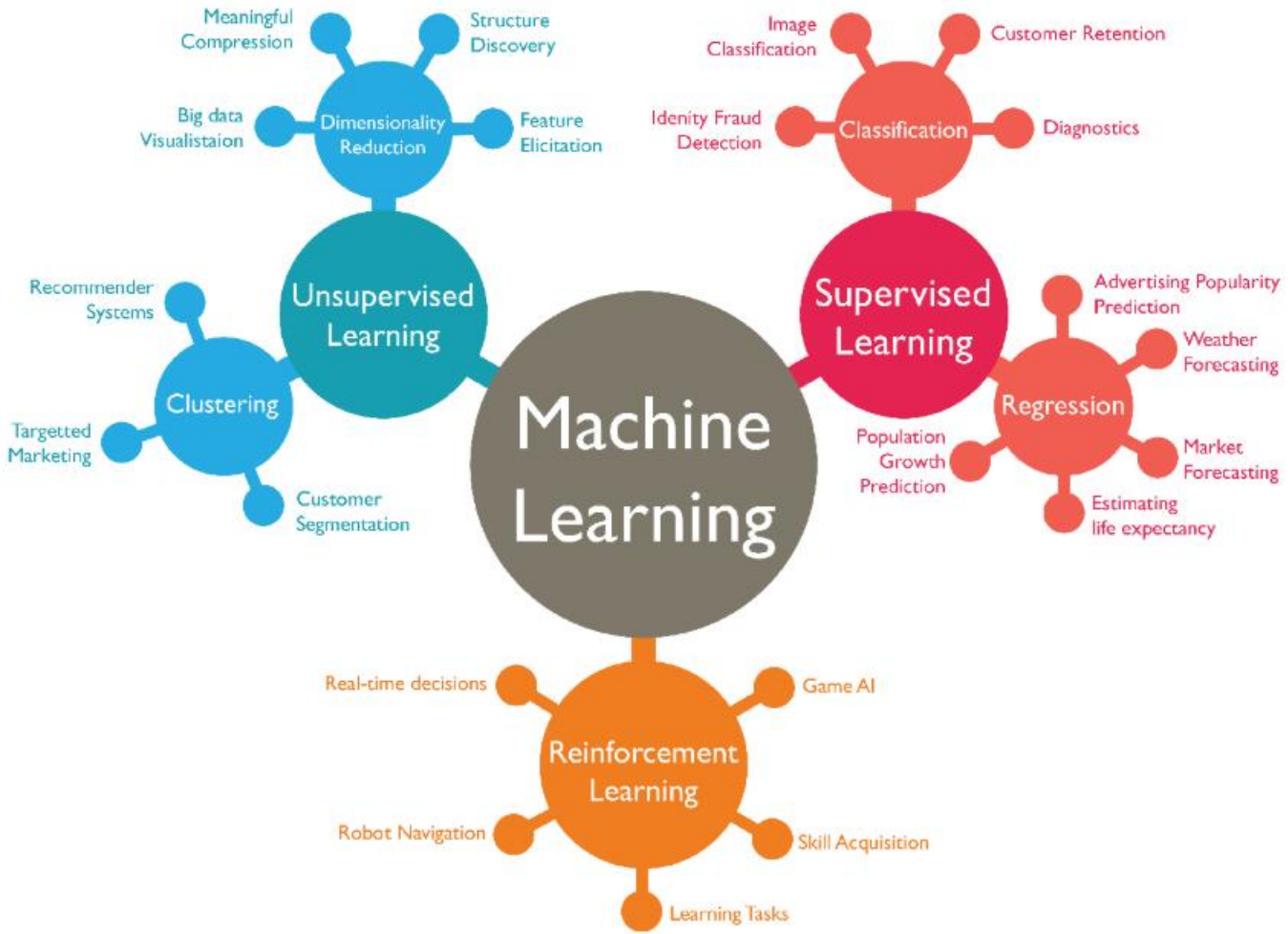
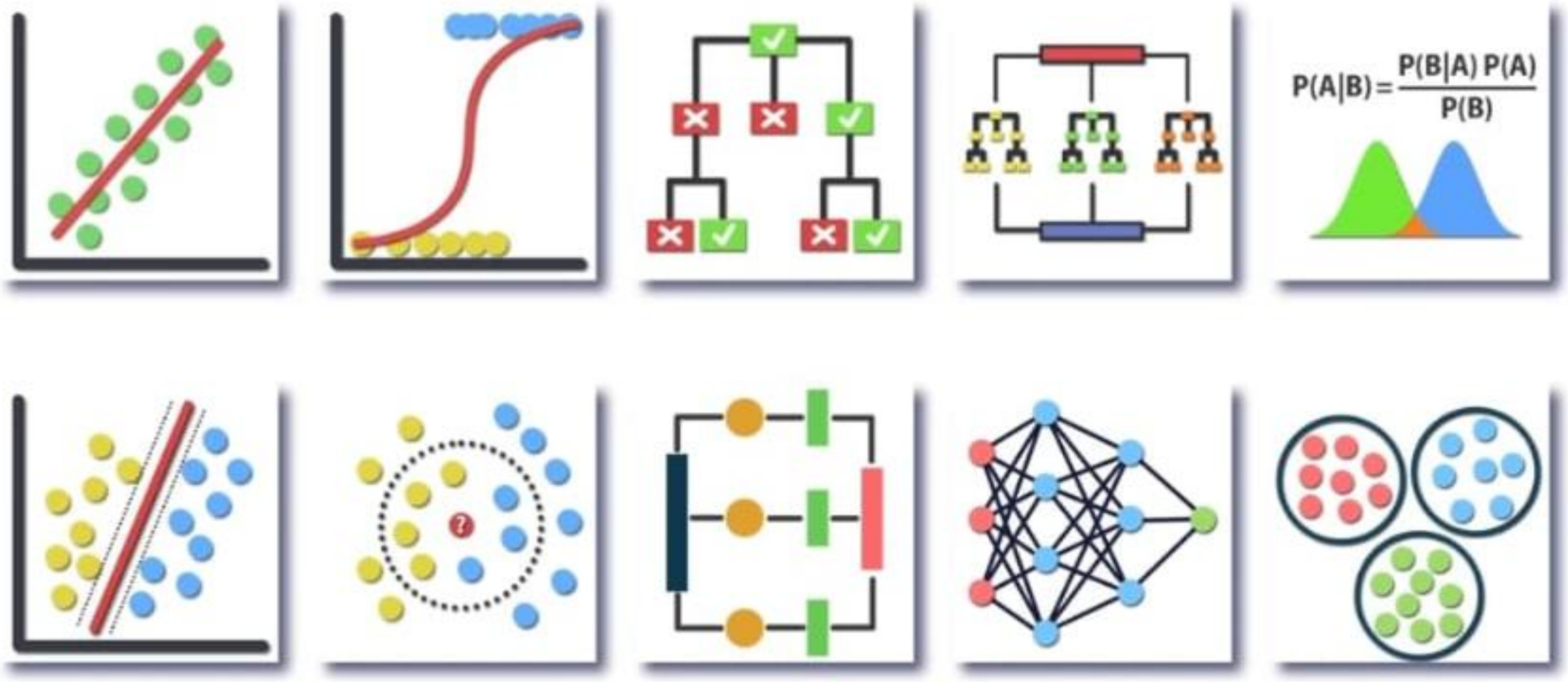
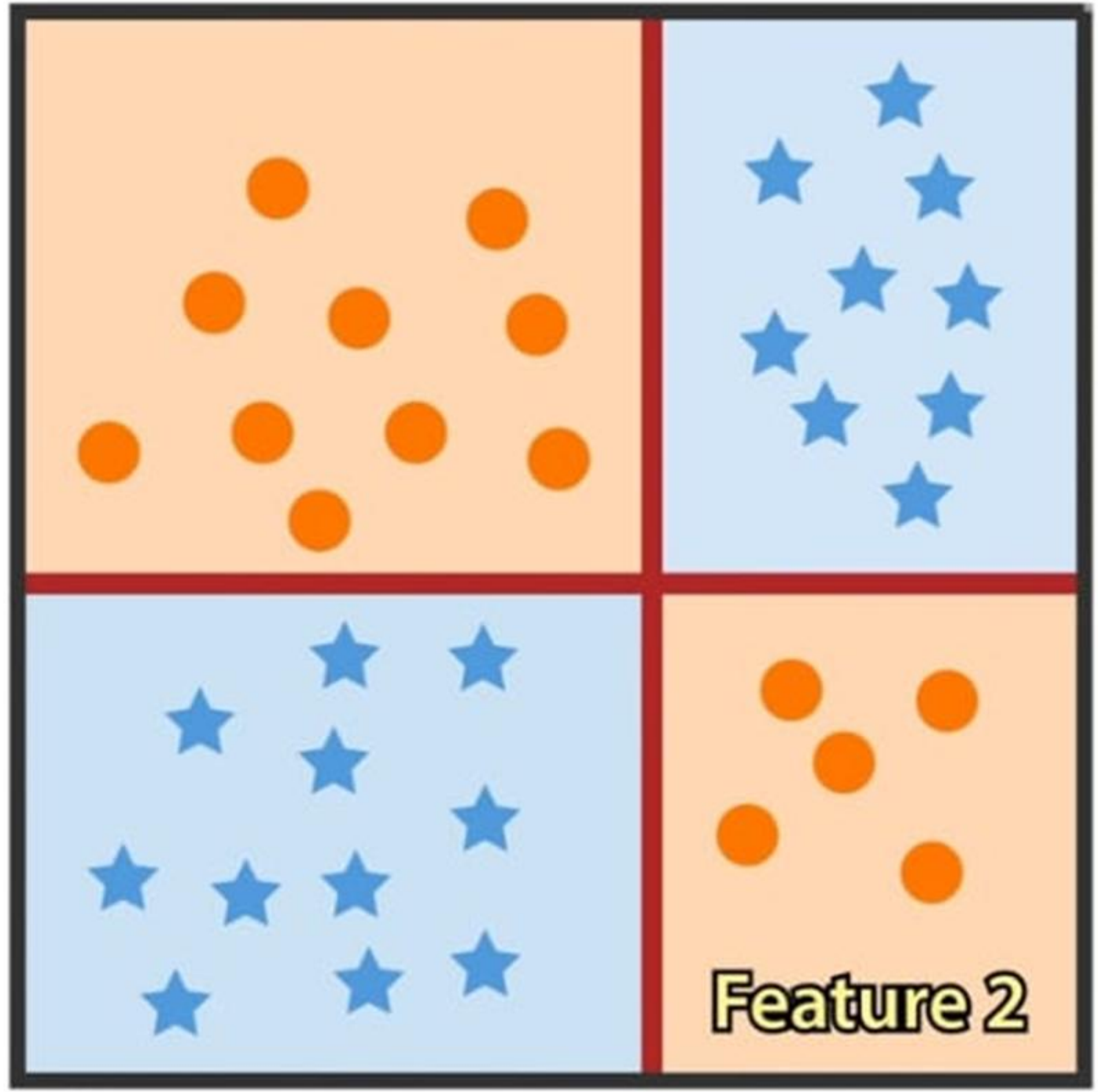


Image via Abdul Rahid

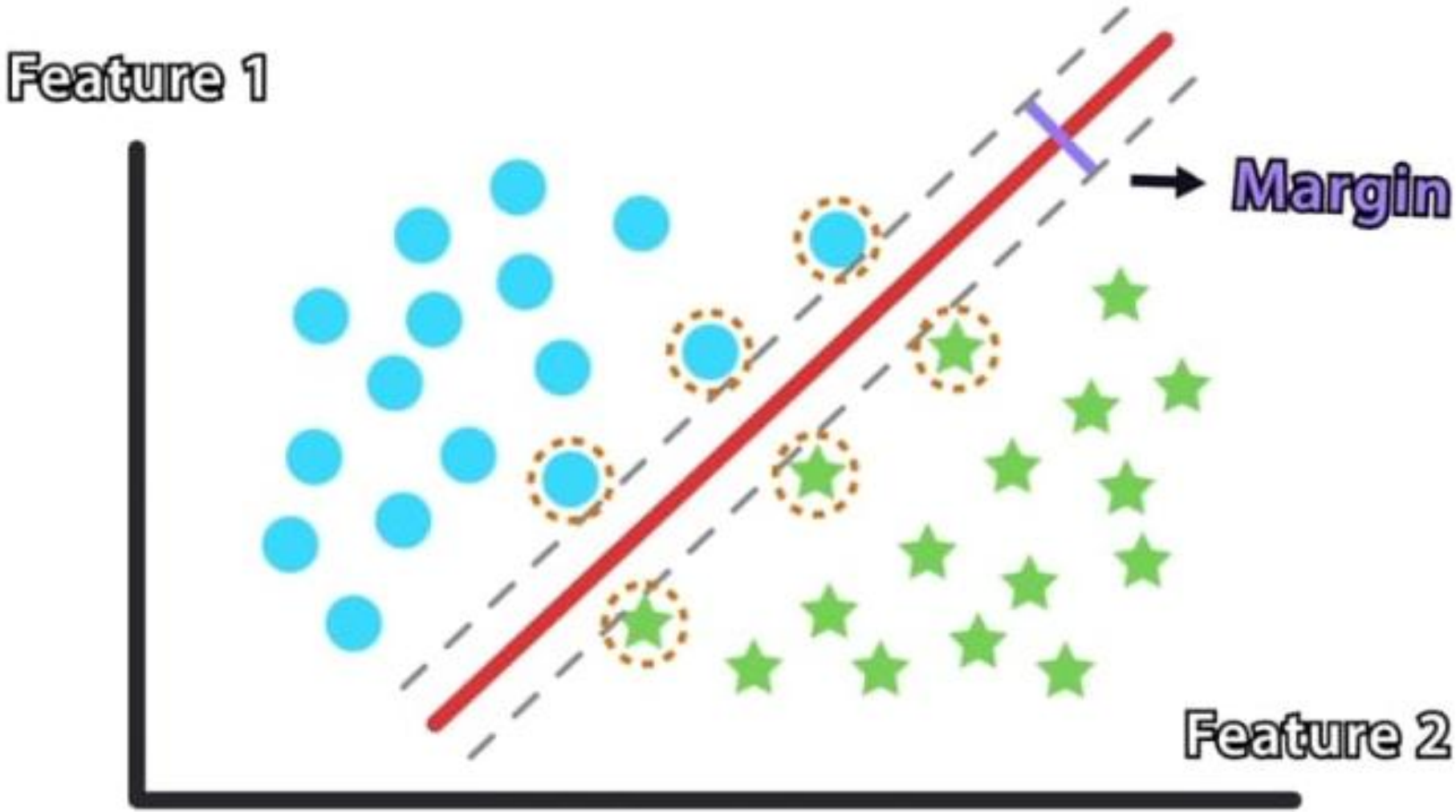
Machine Learning – Algorithms and problems



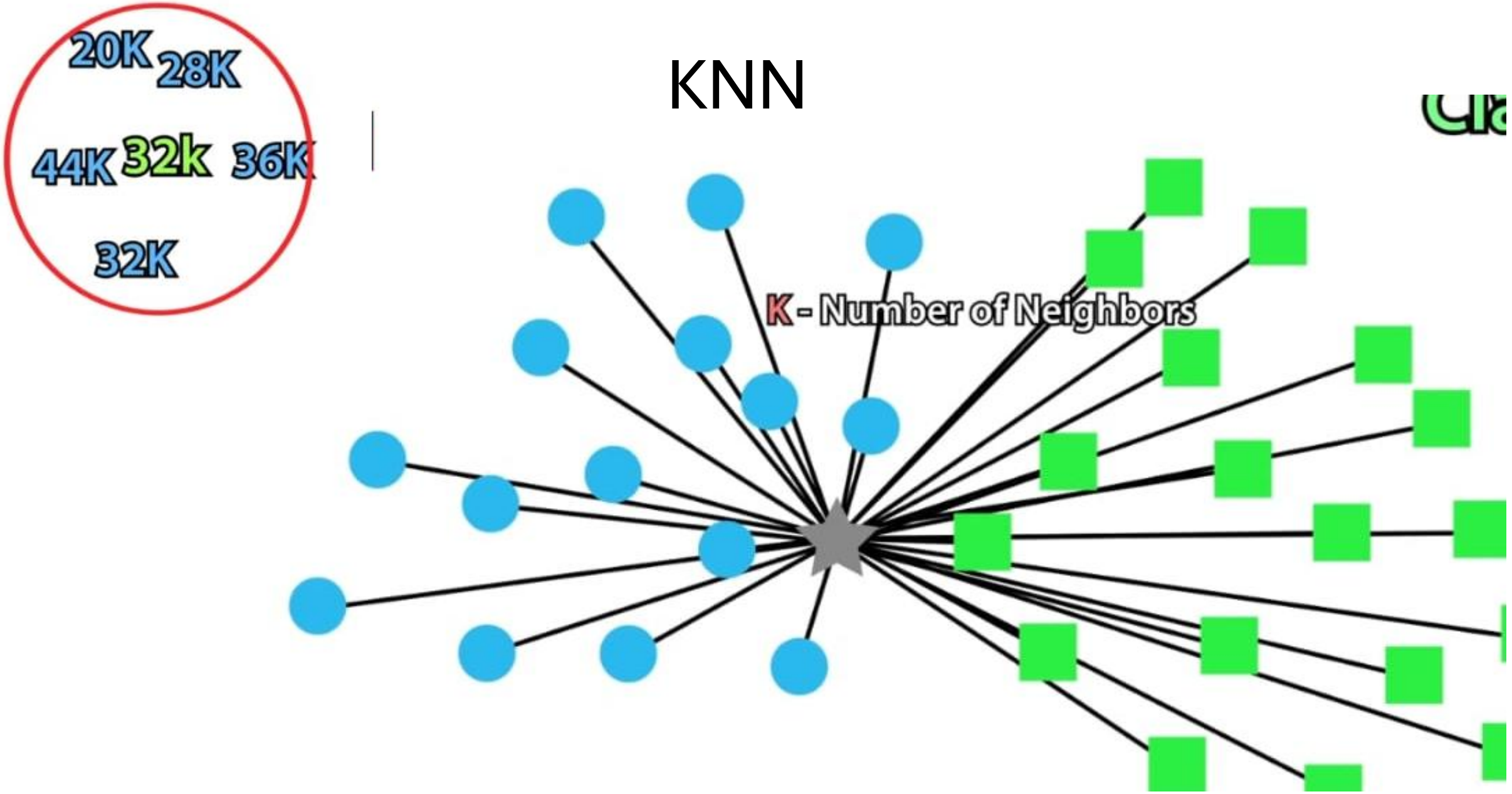
Decision Trees



SVM

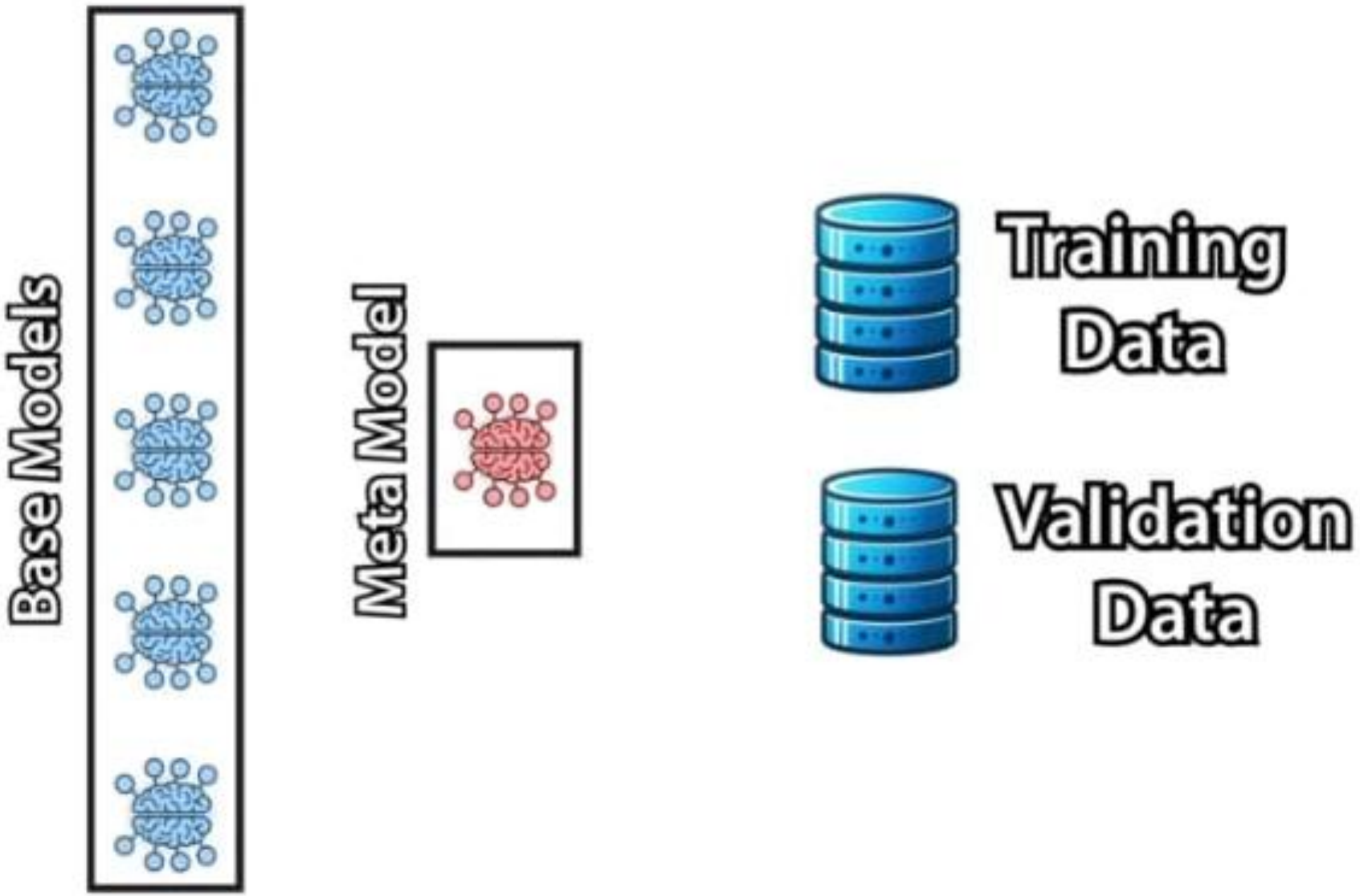


KNN

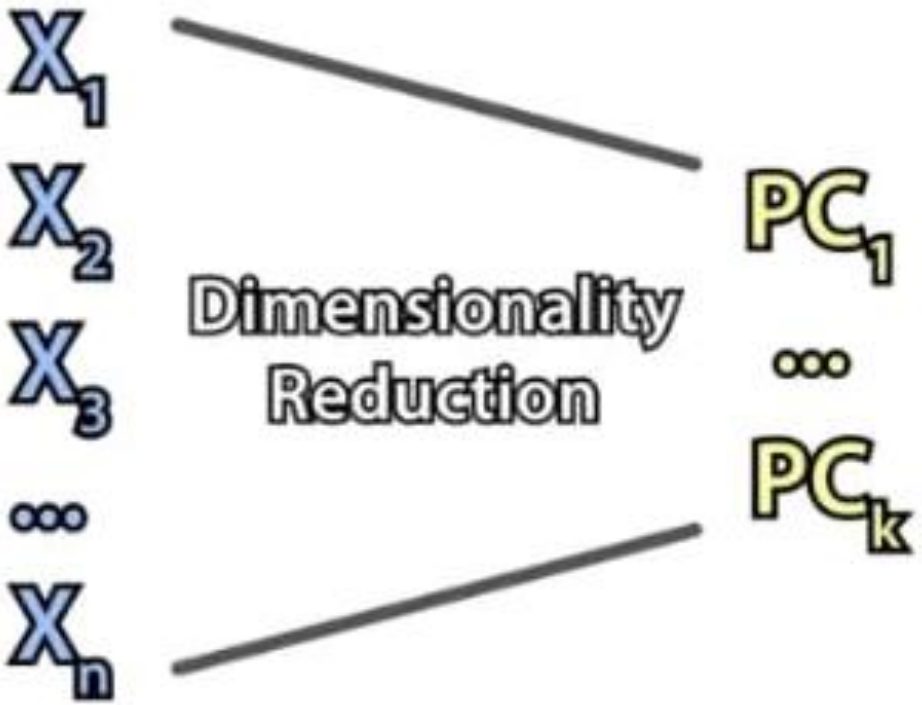


Credit: Easy All For All

Stacking Ensemble Methods



Principal Component Analysis



Where

$k < n$

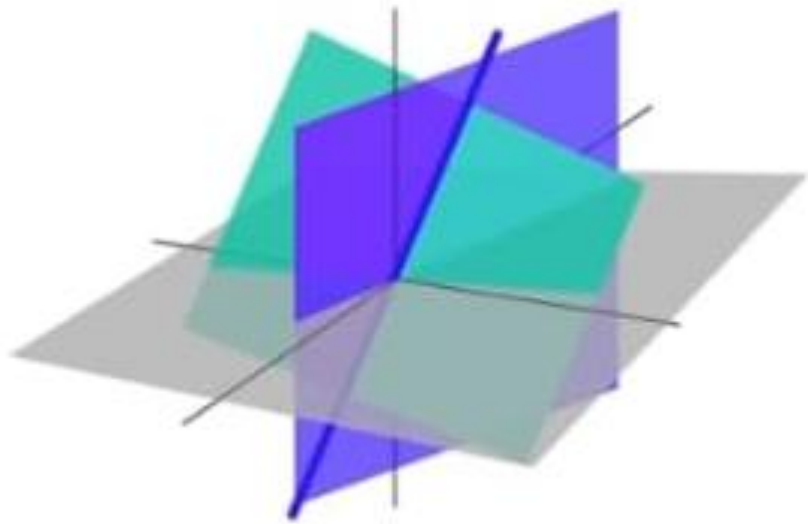
$$PC\,1 = w_1X_1 + w_2X_2 + w_3X_3$$

$$PC\,2 = w_4X_1 + w_5X_2 + w_6X_3$$

Principal Component Analysis

$$PC\,1 = w_1X_1 + w_2X_2 + w_3X_3$$

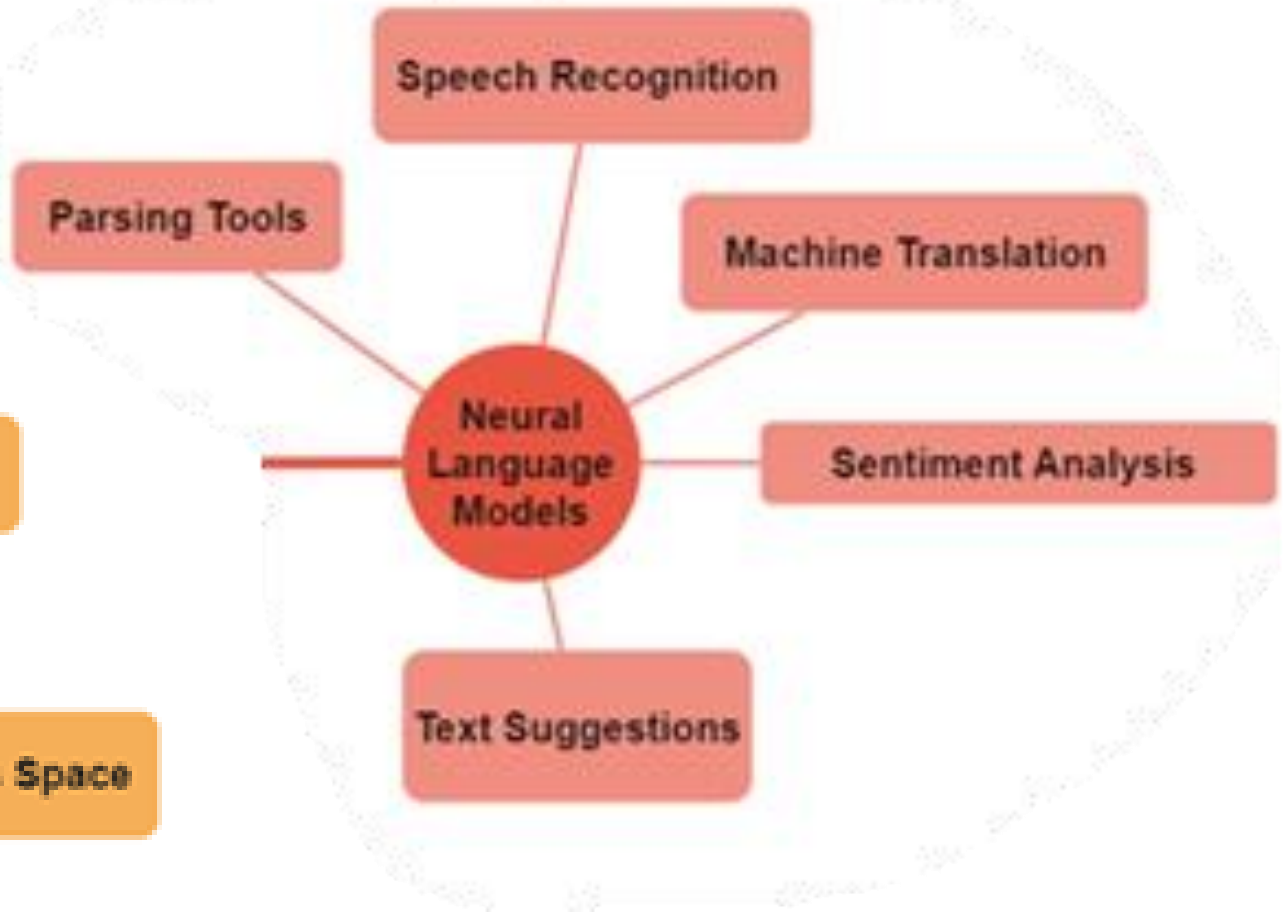
$$PC\,2 = w_4X_1 + w_5X_2 + w_6X_3$$



Eigenvalues and Eigenvectors

$$Av = \lambda v$$

Statistical Language Models & Natural Language Models



Relevancy in IR

- The IR task can be viewed as finding **relevant** documents given a query
  - Main problem – how to model relevancy?

- Relevancy model can be:
  - binary (relevant, non-relevant)
  - probabilistic (score-based)

- Previous probabilistic models try to directly estimate the probability: $\Pr(R \mid query, document)$

Measure the fluency of documents

- How likely this document is generated by a given language model
  - If $p_{text-mining}(d) > p_{health}(d)$, d belongs to text mining related documents
  - If $p_{user_a}(d_1) > p_{user_a}(d_2)$, recommend d_1 to $user_a$
  - A relative concept

What is a Language Model?

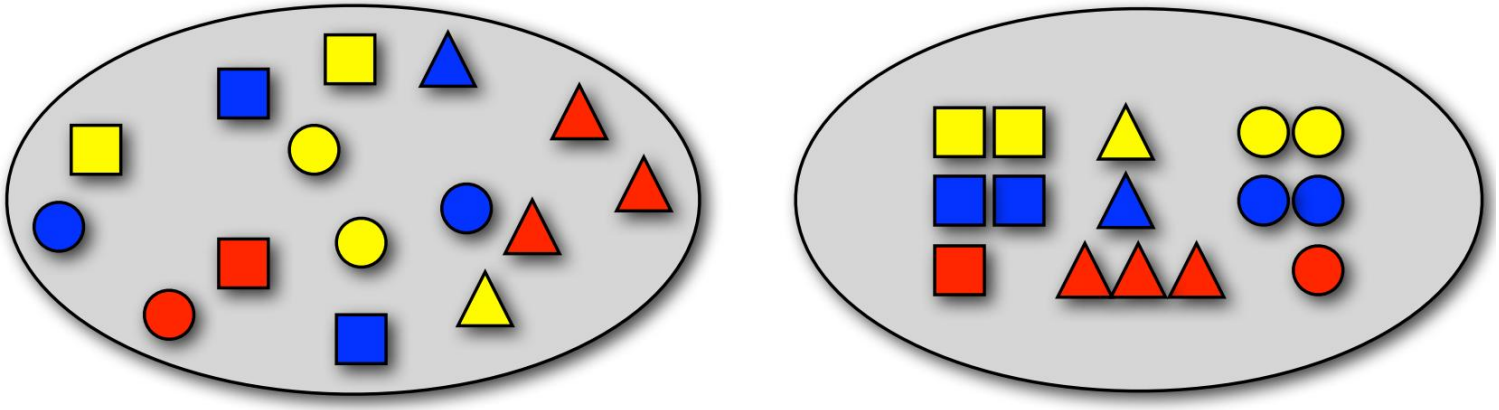
- Probability distribution over strings of text
  - How likely is a given string in a given “language” L
  - Example:
    - $p_1 = Pr_L$(“a quick brown dog”)
    - $p_2 = Pr_L$(“dog quick a brown”)
    - $p_3 = Pr_L$(“grdfgdrfe brown dog”)

    If L is English: $p_1 > p_2 > p_3$

    In most models in this lesson: $p_1 = p_2$ (bag of words)

Sampling with replacement

- Pick a random shape, then put it back in the bag



P(■) = 2/15
P(blue) = 5/15
P(blue |□) = 2/5

P(■) = 1/15
P(red) = 5/15
P(□) = 5/15

P(■ or △) = 2/15
P(△|red) = 3/5

Essential probability concepts

- Probability of events
  - Mutually exclusive events
    - $P(A \cup B) = P(A) + P(B)$
  - General events
    - $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
  - Independent events
    - $P(A \cap B) = P(A)P(B)$

Joint probability, or simply as $P(A, B)$

Why is a LM useful?

- Provide a principled way to quantify the uncertainties associated with natural language
- Allow us to answer questions like:
  - Given that we see "*John*" and "*feels*", how likely will we see "*happy*" as opposed to "*habit*" as the next word?
    (speech recognition)
  - Given that we observe "baseball" three times and "game" once in a news article, how likely is it about "sports" v.s. "politics"?
    (text categorization)
  - Given that a user is interested in sports news, how likely would the user use "baseball" in a query?
    (information retrieval)

Kinds of language models

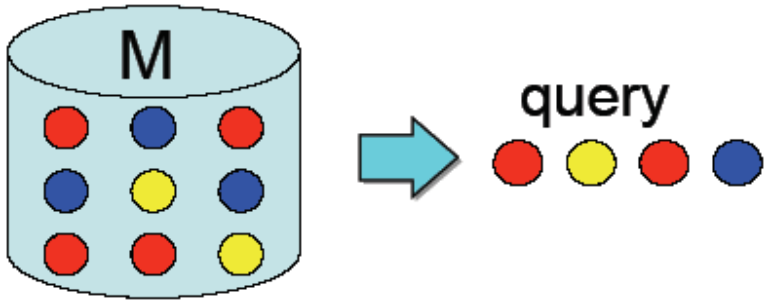
- Linguistic complexity (unigram or higher order)
- Statistical framework (multinomial or multiple-Bernoulli)

What is a statistical LM?

- A model specifying probability distribution over word sequences
  - p("*Today is Wednesday*") ≈ 0.001
  - p("*Today Wednesday is*") ≈ 0.0000000000001
  - p("*The eigenvalue is positive*") ≈ 0.00001
- It can be regarded as a probabilistic mechanism for "generating" text, thus also called a "generative" model

Unigram Language Models

- Words are sampled independently from each other
  - Metaphor: randomly picking words from a bag
  - Independence makes joint probability simple
  - Estimation of probabilities: simple counting



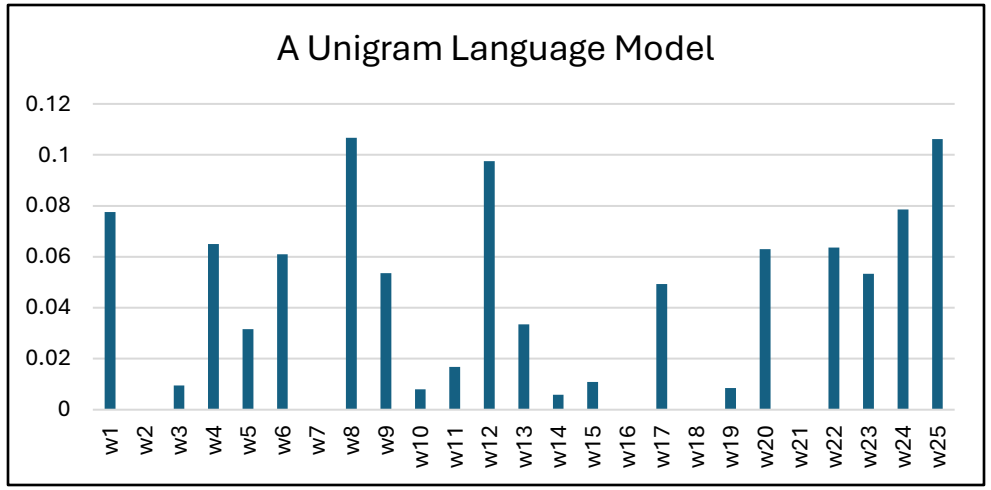
P( ) = P( ) P( ) P( ) P( )

= 4 / 9 * 2 / 9 * 4 / 9 * 3 / 9 = .015

Unigram language model

- Generate a piece of text by generating each word independently
  - $p(w_1\, w_2\, \ldots\, w_n) = p(w_1)p(w_2)\ldots p(w_n)$
  - $s.t. \{p(w_i)\}_{i=1}^{N}, \sum_i p(w_i) = 1, p(w_i) \geq 0$
- Essentially a multinomial distribution over the vocabulary

The simplest and most popular choice!



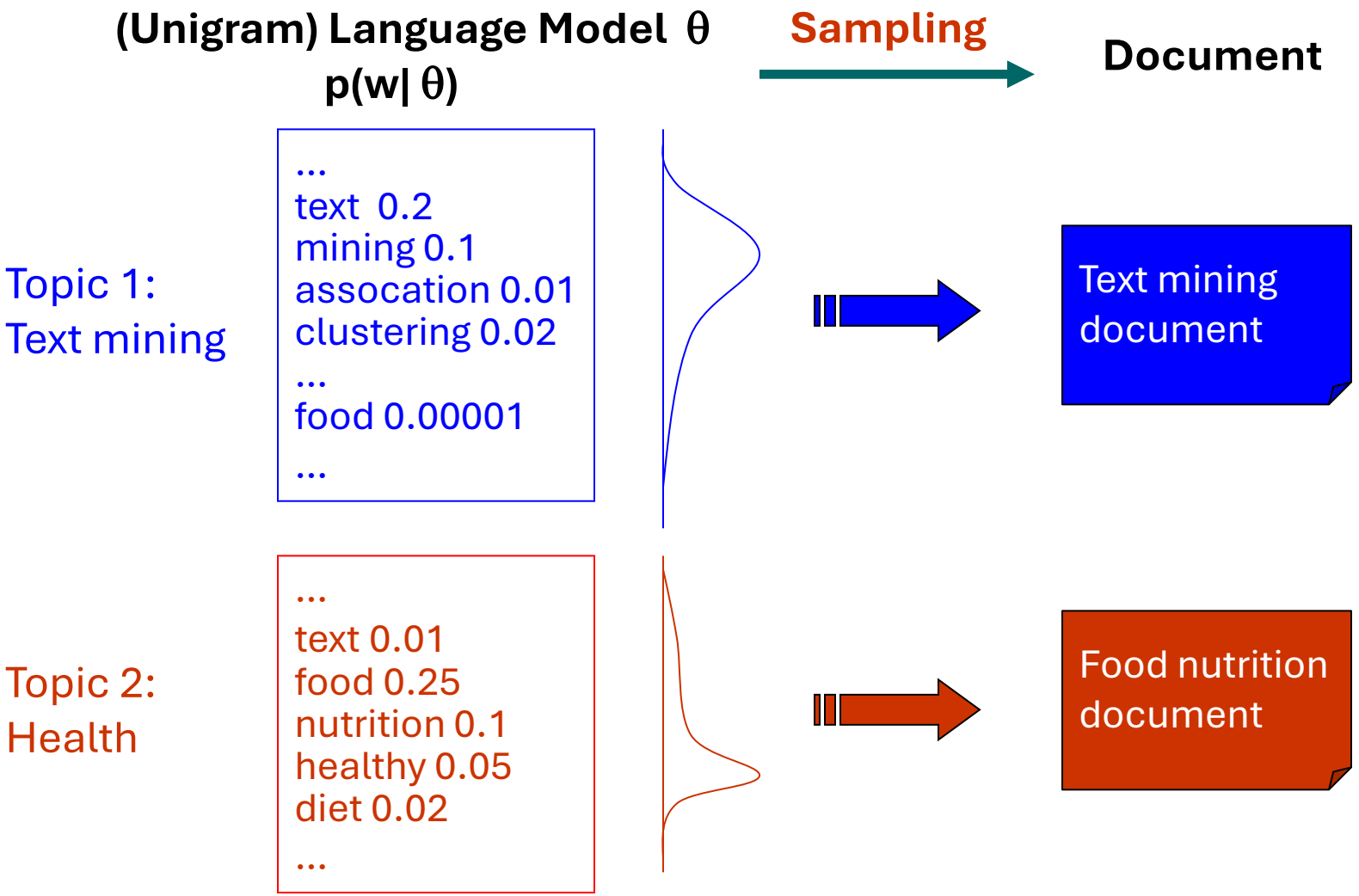
More sophisticated LMs

- N-gram language models
  - Conditioned only on the past n-1 words
  - E.g., bigram: $p(w_1 \ \ldots \ w_n)$
    $= p(w_1)p(w_2|w_1)\ p(w_3|w_2)\ \ldots p(w_n|w_{n-1})$
- Remote-dependence language models (e.g., Maximum Entropy model)
- Structured language models (e.g., probabilistic context-free grammar)

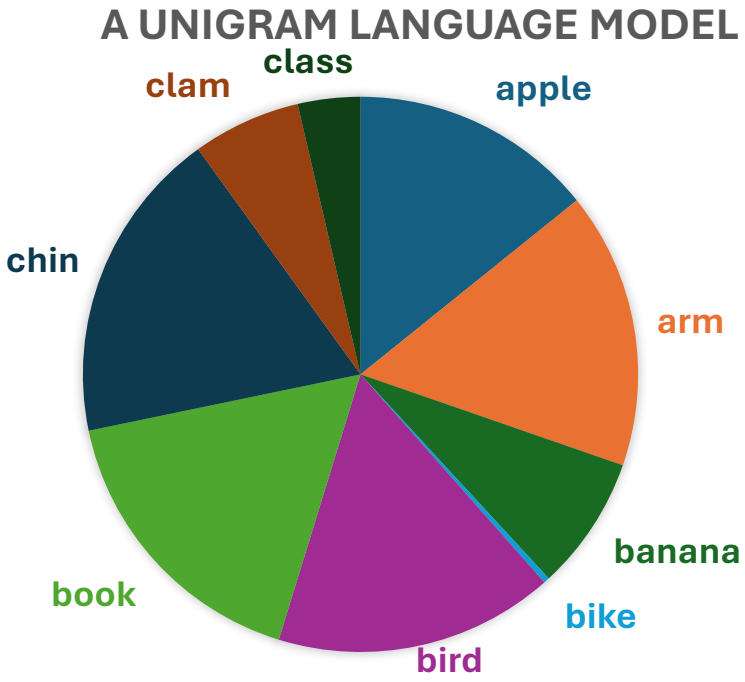
Why just unigram models?

- Difficulty in moving toward more complex models
  - They involve more parameters, so need more data to estimate
  - They increase the computational complexity significantly, both in time and space
- Capturing word order or structure may not add so much value for "topical inference"
- But, using more sophisticated models can still be expected to improve performance ...

Generative view of text documents

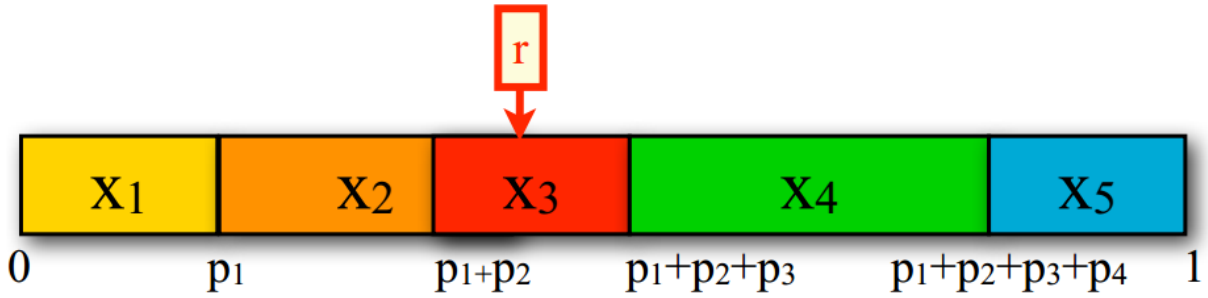


How to generate text from an N-gram language model?



How to generate text from an N-gram language model?

- Sample from a discrete distribution $p(X)$
  - Assume n outcomes in the event space X
  1. Divide the interval [0,1] into n intervals according to the probabilities of the outcomes
  2. Generate a **random** number r between 0 and 1
  3. Return x_i where r falls into



Generating text from language models

P(of) = 3/66
P(Alice) = 2/66
P(was) = 2/66
P(to) = 2/66

P(her) = 2/66
P(sister) = 2/66
P(,) = 4/66
P(') = 4/66

Under a unigram language model:

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

Generating text from language models

P(of) = 3/66
P(Alice) = 2/66
P(was) = 2/66
P(to) = 2/66

P(her) = 2/66
P(sister) = 2/66
P(,) = 4/66
P(') = 4/66

Under a unigram language model:

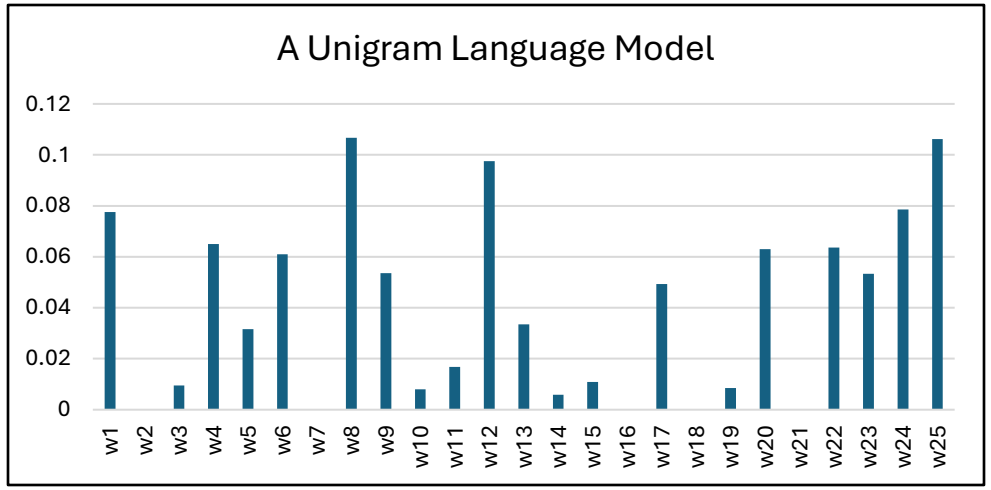
The same likelihood!

beginning by, very Alice but was and?
reading no tired of to into sitting
sister the, bank, and thought of without
her nothing: having conversations Alice
once do or on she it get the book her had
peeped was conversation it pictures or
sister in, 'what is the use had twice of
a book''pictures or' to

Recap: unigram language model

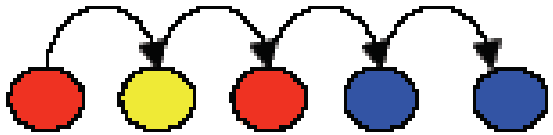
- Generate a piece of text by generating each word <u>independently</u>
  - $p(w_1\, w_2\, \ldots\, w_n) = p(w_1)p(w_2)\ldots p(w_n)$
  - $s.t. \{p(w_i)\}_{i=1}^{N}, \sum_i p(w_i) = 1, p(w_i) \geq 0$
- Essentially a multinomial distribution over the vocabulary

The simplest and most popular choice!

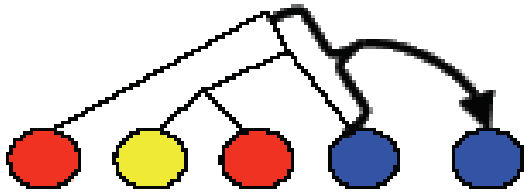


Higher Order Models

- N-gram: condition on preceding words



- Grammar: condition on parse tree



Are they useful?

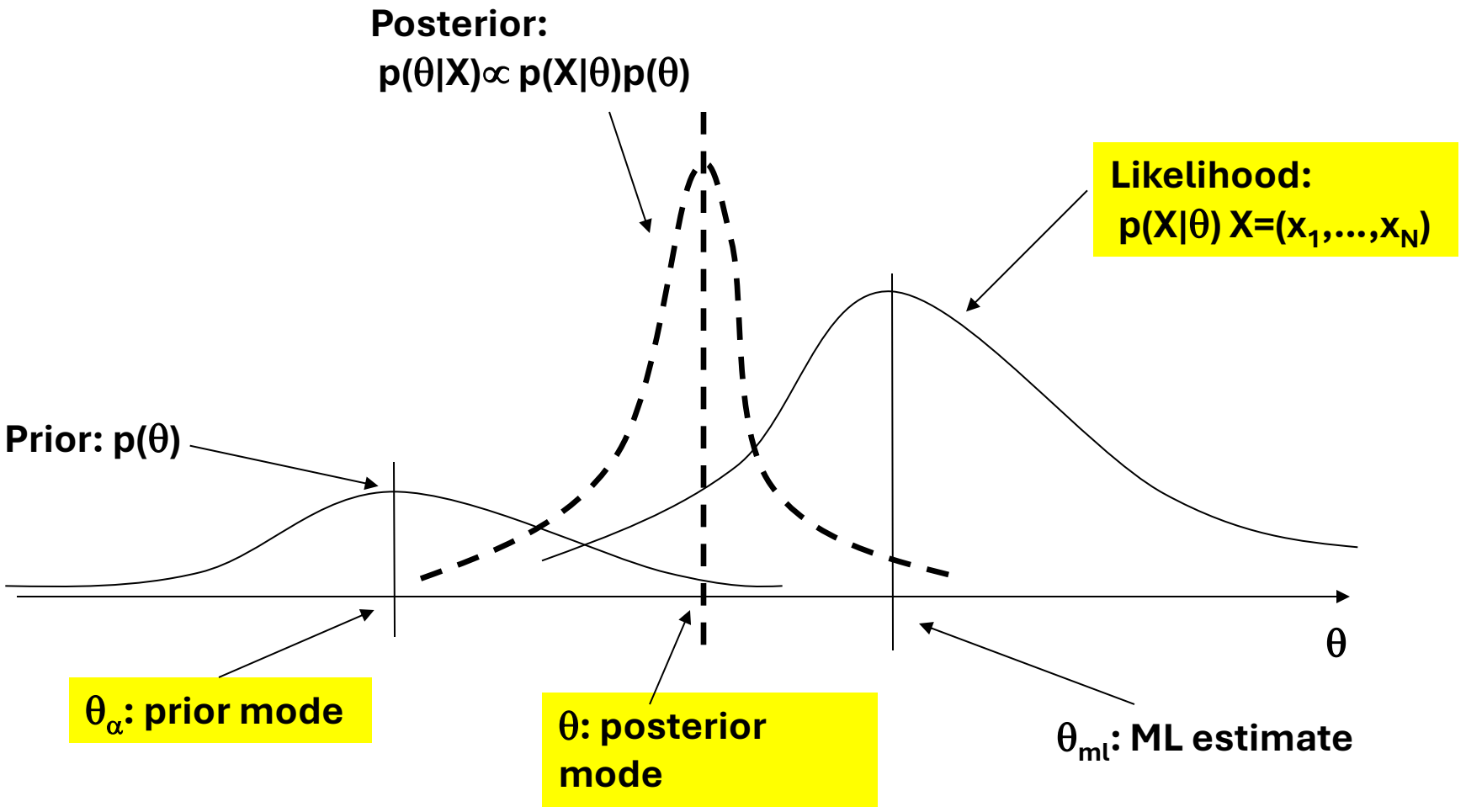
- No Significant improvements from higher models
- Parameter estimation is expensive

N-gram language models will help

Generated from language models of New York Times

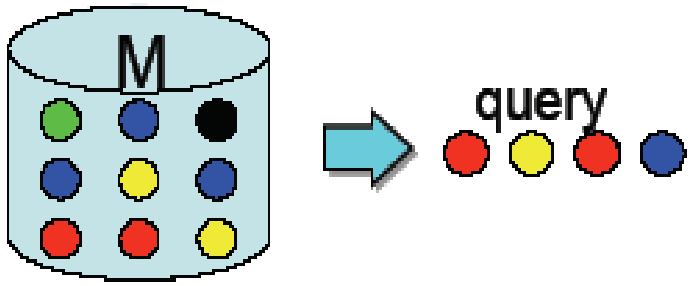
- Unigram
  - Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a q acquire to six executives.
- Bigram
  - Last December through the way to preserve the Hudson corporation N.B.E.C. Taylor would seem to complete the major central planners one point five percent of U.S.E. has already told M.X. corporation of living on information such as more frequently fishing to keep her.
- Trigram
  - They also point to ninety nine point six billon dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions.

Illustration of Bayesian estimation



Multinomial Model (Unigram)

- Fundamental event: what is the identity of the i'th query token?
- Observation is a sequence of events, each for every token



$$\Pr(q_1 \ldots q_n \mid M) = \Pr(\text{len}(q)) = n) \prod_{i=1}^{n} \Pr(q_i \mid M)$$

Multinomial Model (Cont…)

- Predominant model, well understood, lots of research
- Can account for multiple word occurrences in the query
- Same event space as NLP, MT (for integration)

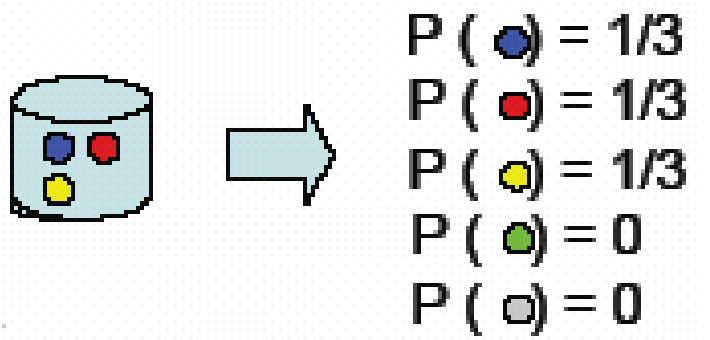
Model Estimation

- General problem:
  - Given a string *S*, estimate its language model M_S
  - S is commonly assumed to be an i.i.d random sample from M_S
- For Unigram we need to estimate $Pr(w \mid M_S)$
- In IR we want to estimate M_q from q and/or M_d from d

Maximum Likelihood

- Count relative frequencies of words in sample *S*:

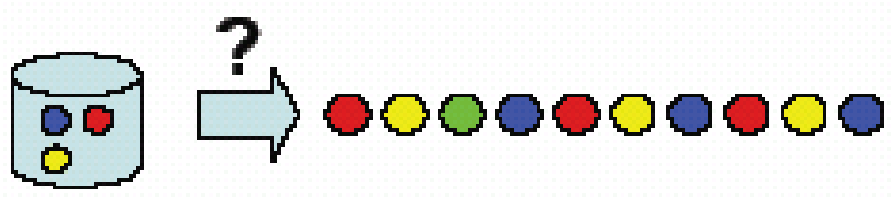
$$\Pr(w \mid M_S) = \frac{C(w,S)}{|S|}$$



- Maximum likelihood property
  - assigns highest possible probability to observation
- Unbiased estimator
  - If we repeat sampling infinite times, we get correct probabilities on average

The Zero Frequency Problem

- Suppose some event is not in observation *S*
  - Model will assign zero probability to the event
  - And to any set of events containing that event
- Happens very frequently with language
- It is incorrect to infer zero probabilities
  - Especially when creating a model from short samples



Discounting Methods

- Laplace correction
  - Add 1 to every count, normalize
  - Problematic for large vocabularies
- Lindstone correction
  - Add a small constant to every count, normalize

$$\Pr(w \mid M_S) = \frac{C(w,S) + \varepsilon}{|S| + \varepsilon |V|}$$

- Absolute discounting
  - subtract constant, redistribute the probability mass

Interpolation Methods

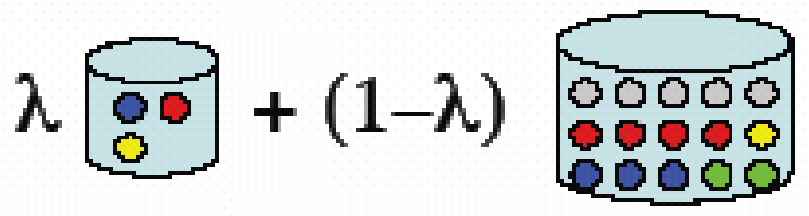
- Problem with discounting methods
  - Discounting treats unseen words equally
  - Some words are frequent than others

- Idea: use background probabilities
  - Interpolate maximum likelihood estimates from S with general English expectations
  - In IR, background probabilities play the role of IDF

Jelinek-Mercer Smoothing

$$\Pr(w \mid M_S) = \lambda P_{ML}(w \mid d) + (1-\lambda) P_{GE}(w)$$

- The same interpolation parameter is used for all documents and queries
- Tune parameter for optimal performance over the document collection



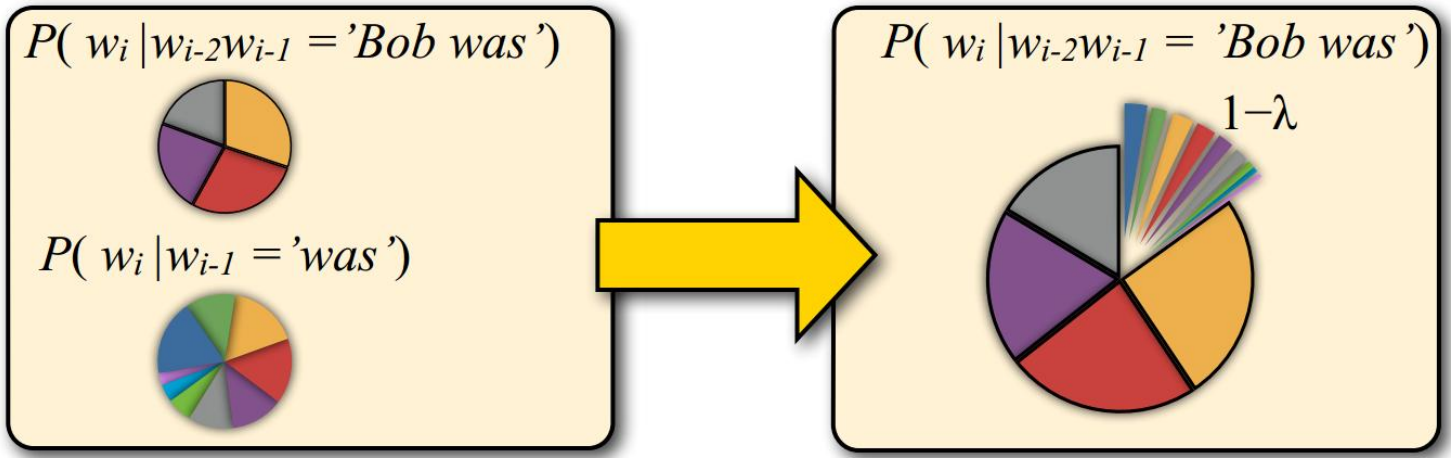
Dirichlet Smoothing

$$\Pr(w \mid M_S) = \frac{C(w,d) + \mu P_{GE}(w)}{|d| + \mu}$$

- The interpolation parameter μ can be viewed as the “size” of a “background document” added to the document
- Longer documents are less smoothed!
- Tune parameter for optimal performance over the document collection

Smoothing methods

- Linear interpolation
  - Use (N –1)-gram probabilities to smooth N-gram probabilities
    - We never see the trigram “Bob was reading”, but we do see the bigram “was reading”

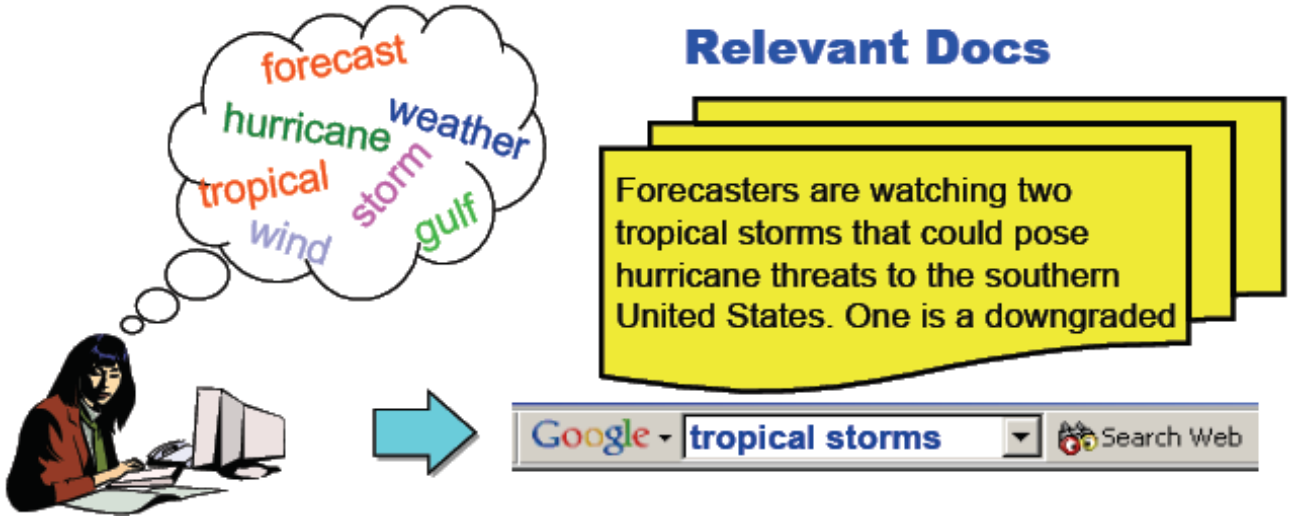


Major Issues in LM for IR

- What kind of language model to use
  - Linguistic complexity (unigram or higher order)
  - Statistical framework (multinomial or multiple-Bernoulli)
- How to estimate model parameters
  - Basic models
  - Translation models
- How to use model for ranking

How can we use Language Models in IR

- Task: given a query, retrieve relevant documents
- Use LM to model document generation
- Use LM to model query generation:
  a. User thinks of some relevant document
  b. User picks some keywords to use as query



Language Modeling for IR

- Every document *d* in a collection is generated by a language model M_d
  - Consider the language model as all possible strings that author could have written down when creating a document
  - For a specific author writing a specific document, some strings are more likely to occur than others
    - Topic, writing style, language (english, french…)
  - $Pr(S \mid M_d)$ – the probability that author would write down string *S*

Language Modeling for IR (Cont…)

- Given a user query q, what is the probability that author would write down q?
  - Higher probability captures a notion of relevancy of the document to the query

- Rank documents in a collection by $Pr(q \mid M_d)$
  - Probability of observing q during random sampling from the language model of the document

Ranking with Language Models

- Standard approach – **query likelihood**:
  - Estimate a language model M_d for every document d in the collection
  - Rank documents by probability of "generating" the query:

$$\Pr(q_1 \ldots q_n \mid M_d) = \prod_{i=1}^{n} \Pr(q_i \mid M_d)$$

Drawbacks of Query Likelihood

- No notion of relevance (everything is random sampling)

- User feedback and query expansion not part of the model
    - Examples of relevant documents cannot help improve M_d
    - Augmenting only by adding extra terms to the query

Ranking with Document Likelihood

- Flip the direction of the query-likelihood approach:
  - Estimate a language model M_q for every query q
  - M_q expected to "predict" a typical relevant document
  - Rank documents by the likelihood of being a random sample from M_q:

$$\Pr(d \mid M_q) = \prod_{w \in d} \Pr(w \mid M_q)$$

Problems of Document Likelihood

- Different document lengths, probabilities not comparable
- Favors documents that contain frequent (low content) words
- Consider ideal ("highest ranking") document for a given query and a fixed document length n:

$$\max_d \prod_{w \in d} \Pr(w \mid M_q) = \max_w \Pr(w \mid M_q)^n$$

Ranking with Model Comparison

- Combine advantages of two ranking methods:
  - Estimate a model for both the document and the query
  - Directly compare similarity of the two models
- Natural measure of similarity is cross-entropy (others exist):

$$H(M_q \,||\, M_d) = \sum_{w \in V} \Pr(w \mid M_q) \log(\Pr(w \mid M_d))$$

Ranking with Cross-Entropy

$$H(M_q \,||\, M_d) = \sum_{w \in V} \Pr(w \mid M_q) \log(\Pr(w \mid M_d))$$

- Number of bits we need to "encode" M_q using M_d
- Equivalent to KL divergence: $H(M_q || M_d) - H(M_q)$
- Equivalent to query likelihood if M_q is simply counts of words in q
- Cross-entropy is **not** symmetric

Variability problem with LM

- Basic language models do not address word synonymy
  - If a query q mentions ‘cars’, M_q should have high probability for ‘vehicles’, ‘autos’

- The same problem for cross-lingual retrieval
  - When searching for ‘cars’ in Spanish documents, it may be interesting to see documents containing ‘carros’ or ‘coches’

Translation Models

- Sample a word w from language model M with probability P(w|M)
  - Any of the models we have seen so far
- Translate w into a word v with probability T(v|w)

$$\Pr_{trans}(v \mid M) = \sum_{w \in V} T(v \mid w) \Pr(w \mid M)$$

- Translation probability reflects relationship between v and w

Using Translation Models

- Mono-lingual retrieval
  - v and w are synonyms
  - v and w are topically related, T(v|w) is given by co-occurrence
    - T(v|w) high if v is likely to appear in documents containing w
- Stemming
  - v and w are in the same stem class
- Cross-lingual retrieval
  - v is the translation of w into a different language

Perplexity

- Standard evaluation metric for language models
  - A function of the probability that a language model assigns to a data set
  - Rooted in the notion of cross-entropy in information theory

Perplexity

- The inverse of the likelihood of the test set as assigned by the language model, normalized by the number of words

$$PP(w_1, \dots, w_N) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} p(w_i | w_{i-1}, \dots, w_{i-n+1})}}$$

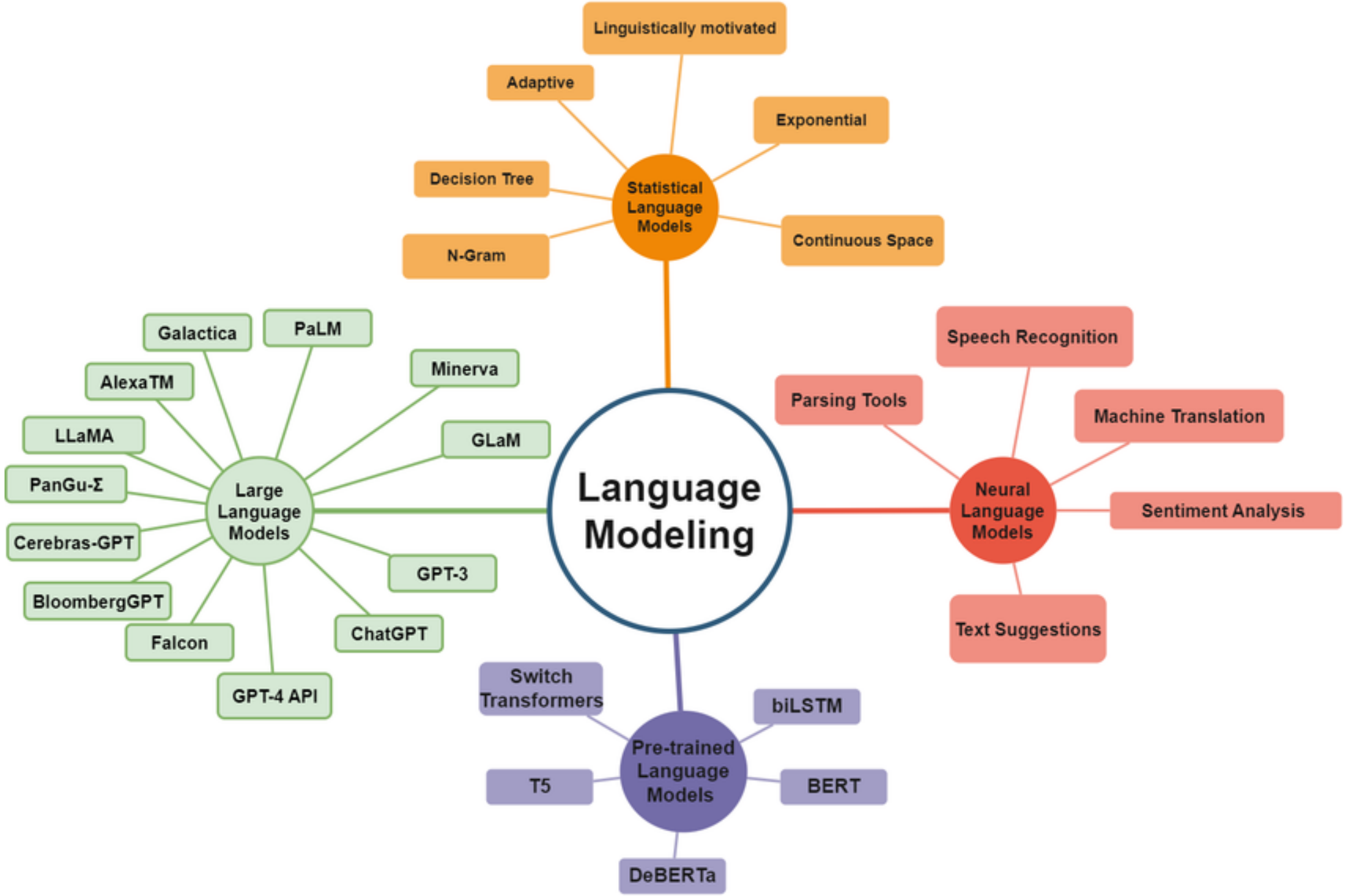
N-gram language model

Common Practice

- What is the main-stream:
  - Unigram models
  - Multinomial models
  - Model estimation by MLE smoothed by interpolation with background probability
  - Ranking by Model comparison (cross-entropy)

Language Modeling: pros & cons

- Pros:
  - Formal mathematical model
  - Simple, well-understood framework
  - Integrates both indexing and retrieval models
  - Natural use of statistics, no heuristics
  - Avoids problematic issues of “relevance”
- Cons:
  - Difficult to incorporate notions of “relevance” and user preference
  - Relevance feedback and query expansion is not straightforward
  - Accommodating phrases, paragraphs and Boolean operators not straightforward


Language Modeling
Statistical Language Models
Linguistically motivated
Adaptive
Exponential
Decision Tree
Continuous Space
N-Gram
Large Language Models
Galactica
PaLM
Minerva
AlexaTM
LLaMA
GLaM
PanGu-Σ
Cerebras-GPT
GPT-3
BloombergGPT
ChatGPT
Falcon
GPT-4 API
Neural Language Models
Speech Recognition
Parsing Tools
Machine Translation
Sentiment Analysis
Text Suggestions
Pre-trained Language Models
Switch Transformers
biLSTM
T5
BERT
DeBERTa

One-Hot Encoding?

- If we have a sequence of T words, and each word gets a one-hot encoded vector of size V, then one sentence becomes a T x V matrix
- Same generic shape as a T x D matrix (as you saw previously)
- E.g. “I like cats”
- Might become: [[0, 0, 0, 1], [0, 1, 0, 0], [1, 0, 0, 0]]
- (If we had a 4-word vocabulary)

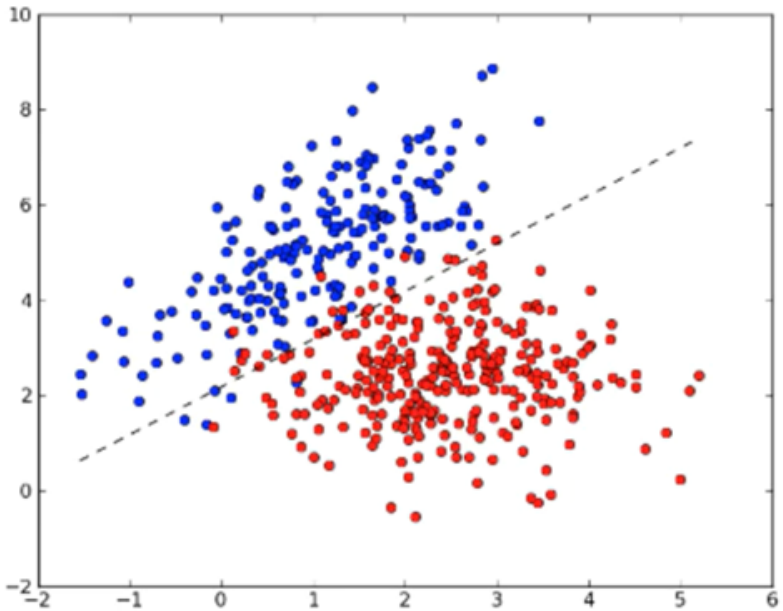
Problem!

- Some English-language datasets have ~1 million possible tokens / words
- This means V (or equivalently, D) is 1 million
- Not only is the input large, but the input-hidden weight matrix will also be large!
- In actuality, the English dictionary has ~200,000 words, but this is still a very large number

	o_ENE	o_ESE	o_East	o_NE	o_NNE	o_NNW	o_NW	o_North	o_SE	o_SSE	o_SSW	o_SW	o_South	o_Variable	o_WSW
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

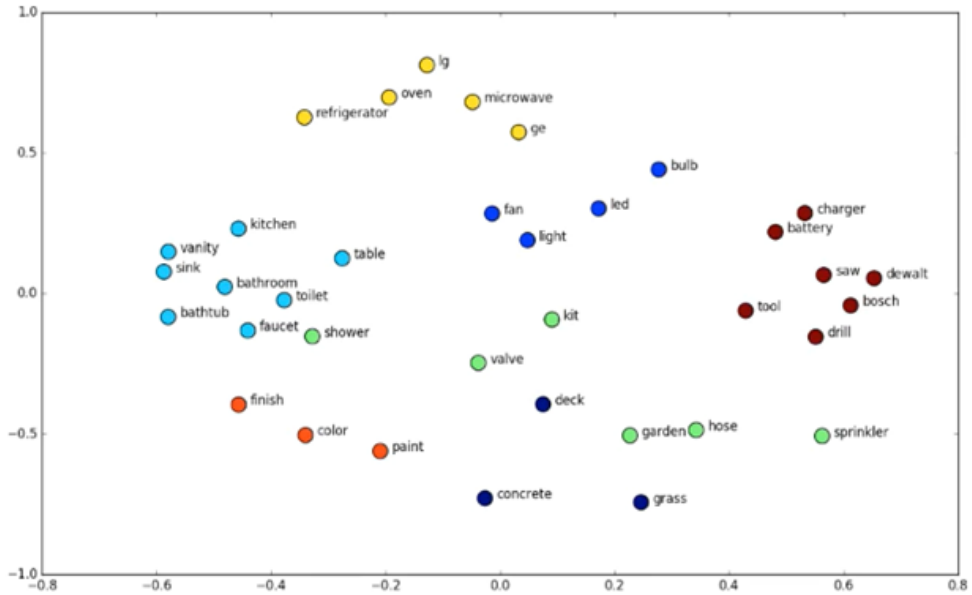
Another problem!

- Recall: we want our input features to have some structure
- “Machine learning is nothing but a geometry problem”
- This rule **relies** on the fact that there’s some geometrical structure in the data - that data vectors in the same class (e.g. spam vs. not spam) are *probably* near each other



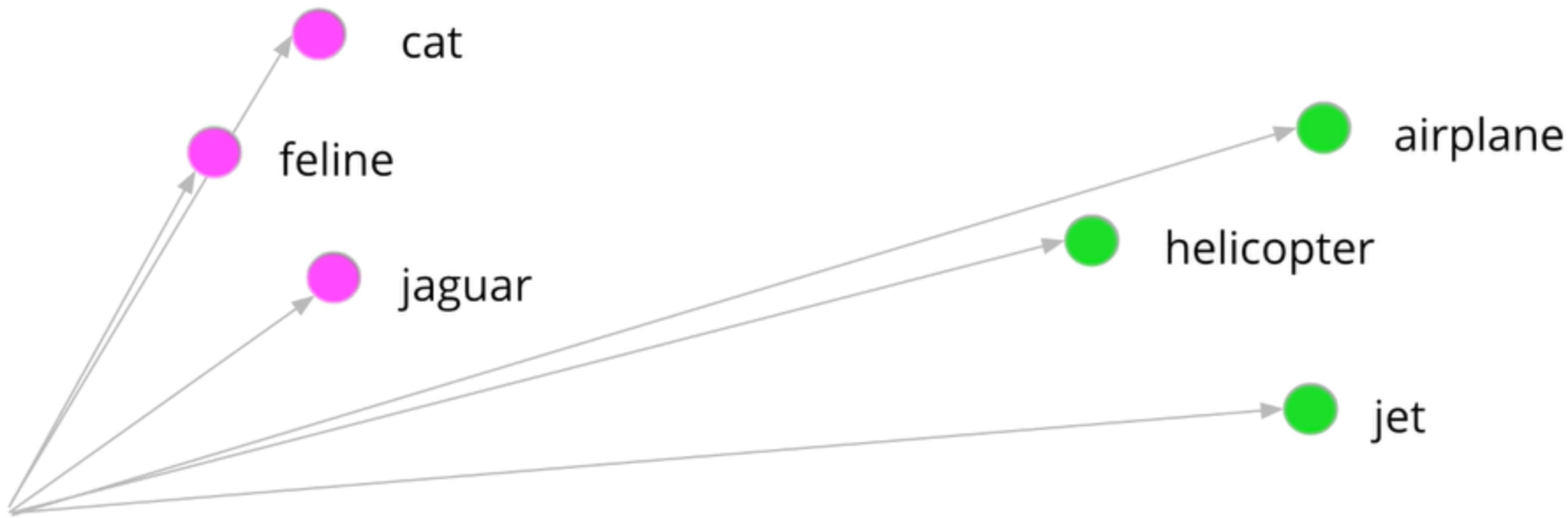
Problem with one-hot encoding

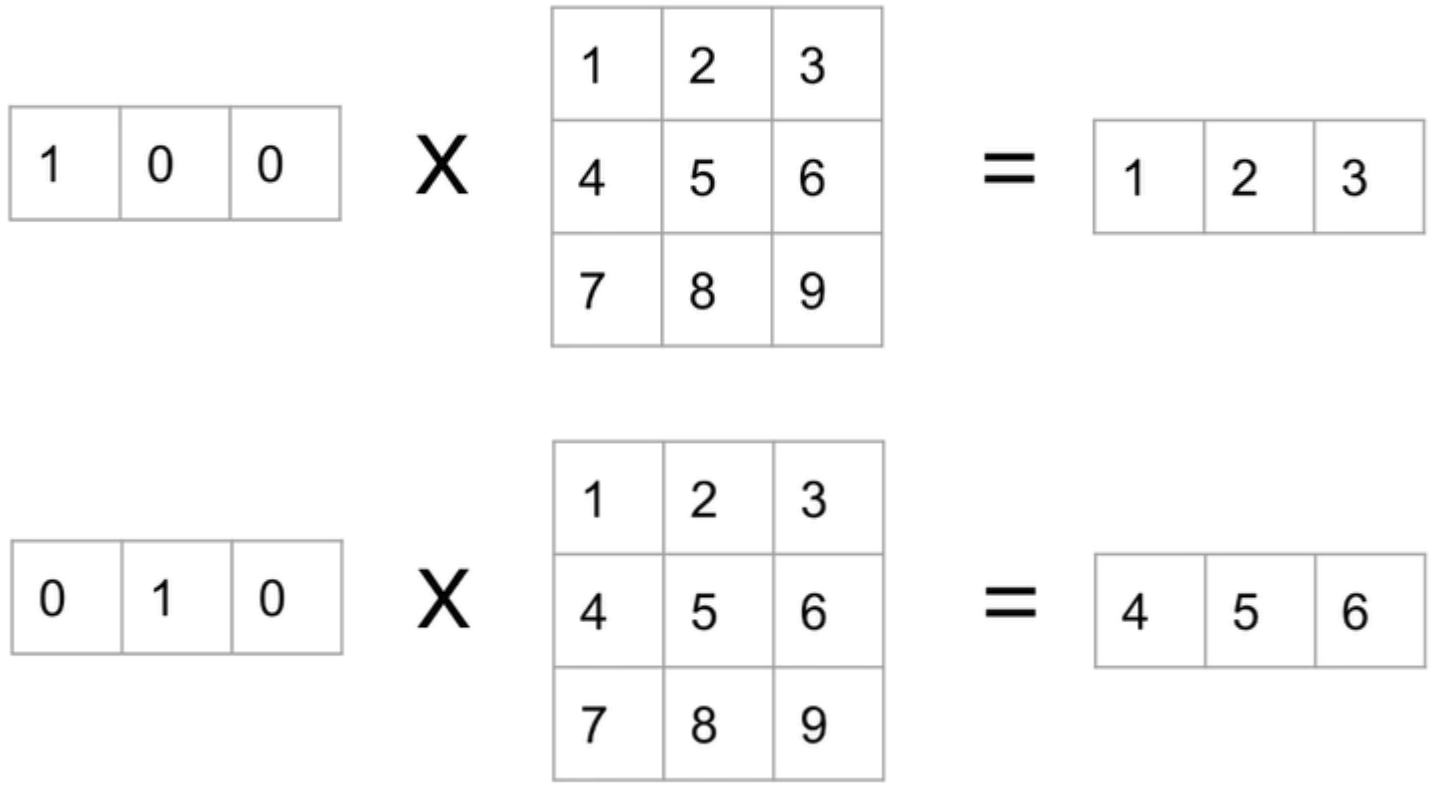
- Take any two one-hot encoded vectors, e.g. [1, 0, 0] vs. [0, 1, 0]
- The Euclidean distance is sqrt($1^2 + 1^2$) = sqrt(2)!
- All words therefore are an equal distance apart!
- The distance between “cat” and “feline” is sqrt(2)
- The distance between “cat” and “airplane” is *also* sqrt(2)
- Useless!



A better solution: Embeddings

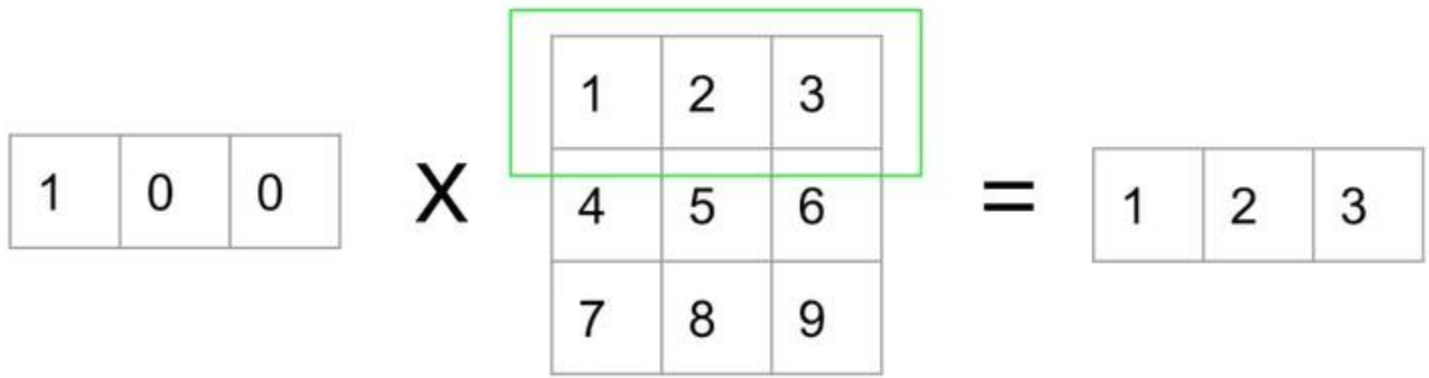
- Assign each word to a D-dimensional vector (not one-hot encoded)
- How are these vectors “found”? We will see shortly!

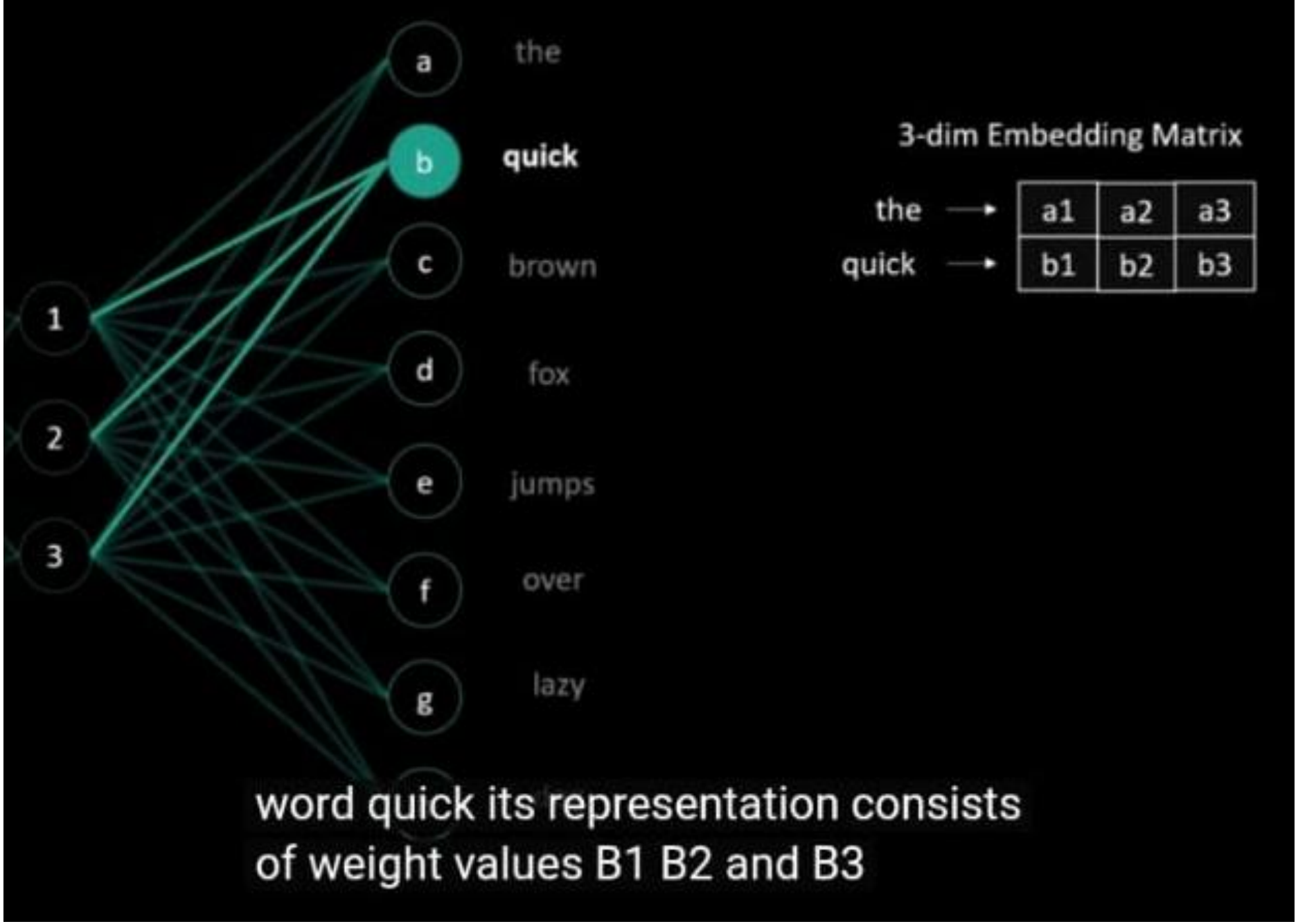



1 0 0
X
1 2 3
4 5 6
7 8 9
=
1 2 3
0 1 0
X
1 2 3
4 5 6
7 8 9
=
4 5 6

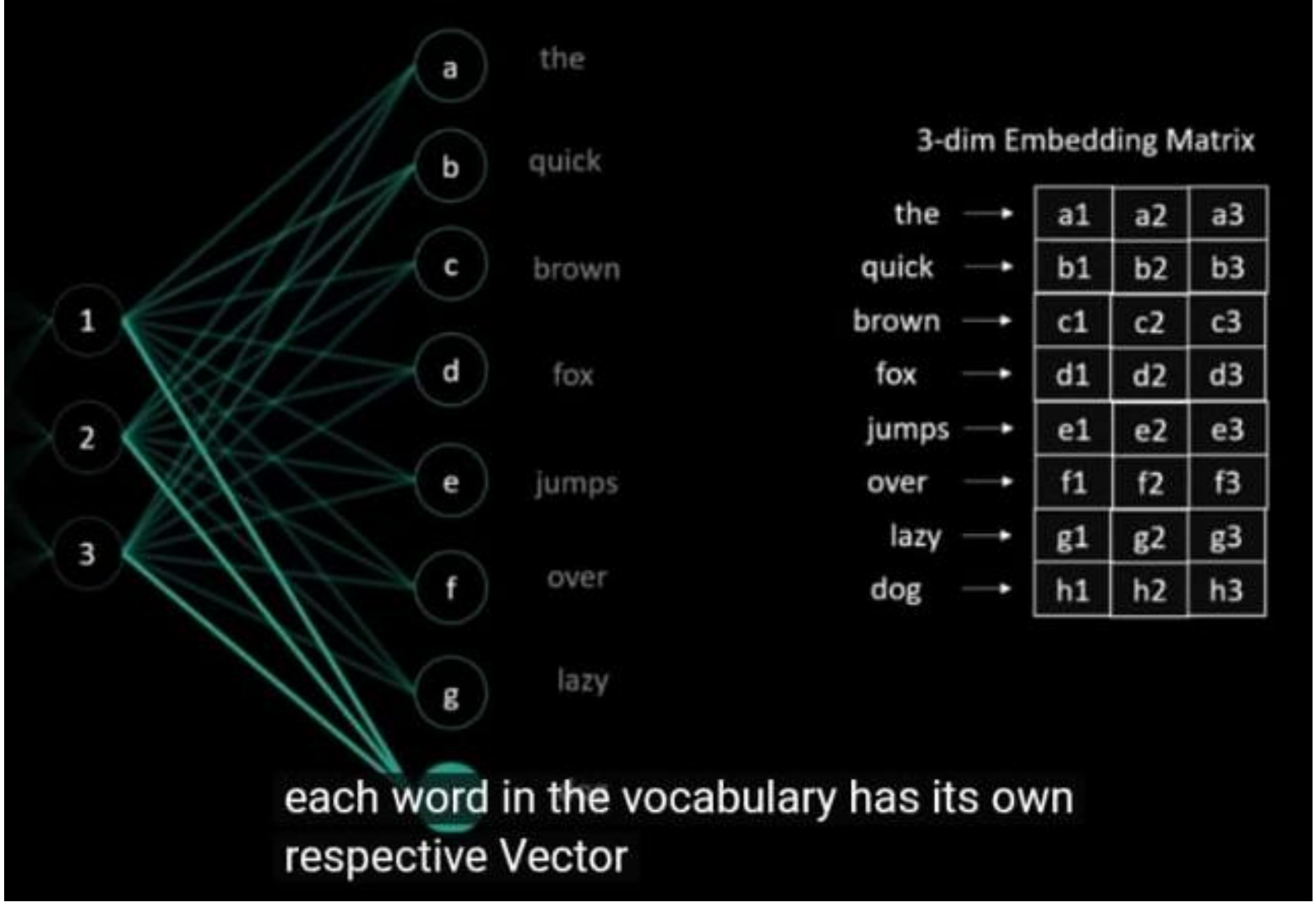
Why is it more efficient?

- Old way takes 2 steps:
  - Create a vector of size V containing all zeros, set the k'th entry to 1
  - Multiply the one hot vector by the weight matrix
- New way takes 1 step:
  - Index the weight matrix (W[k])

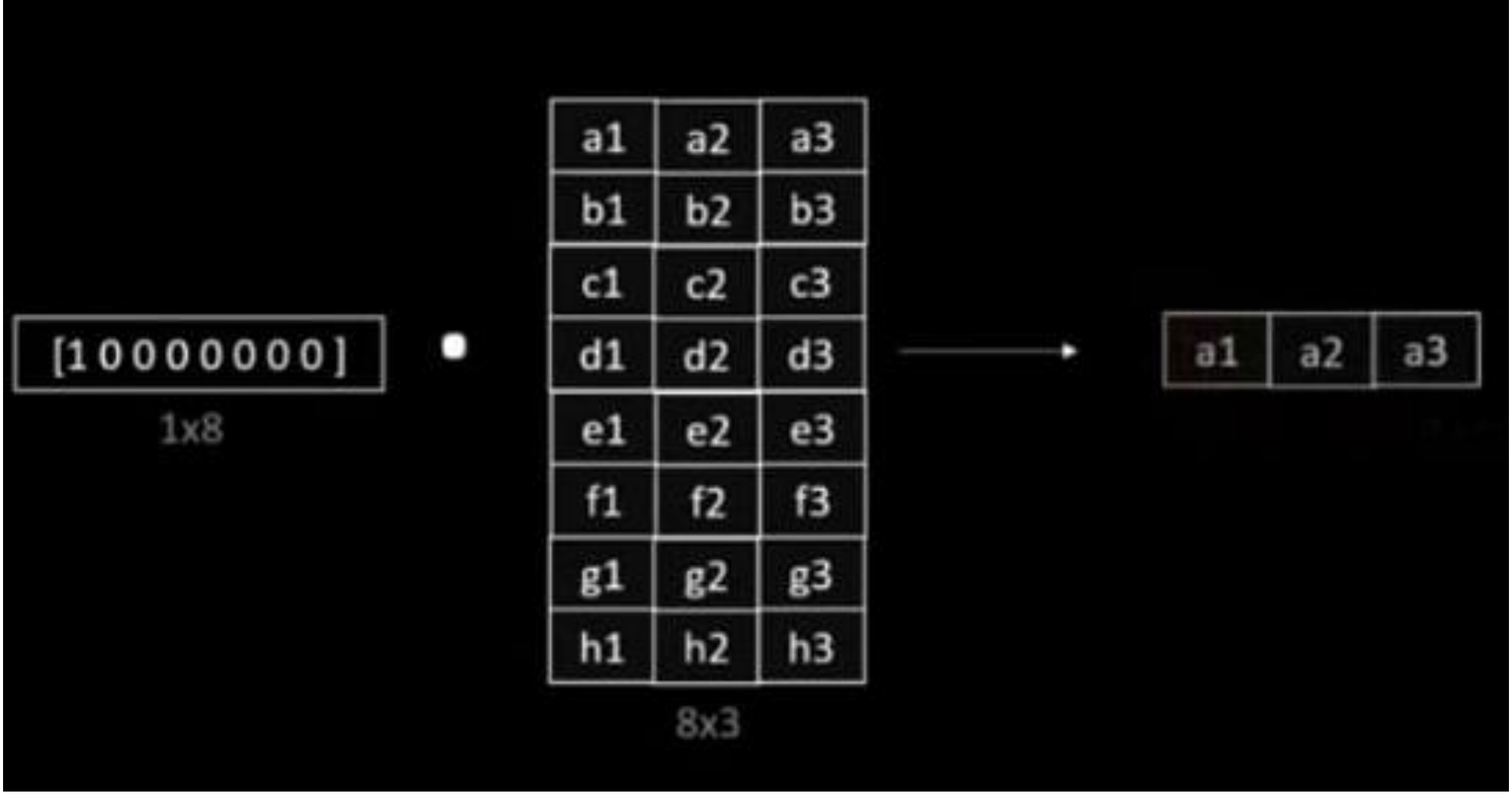




Credit: AbsolutelyAwesome (@Youtube)



Credit: AbsolutelyAwesome (@Youtube)



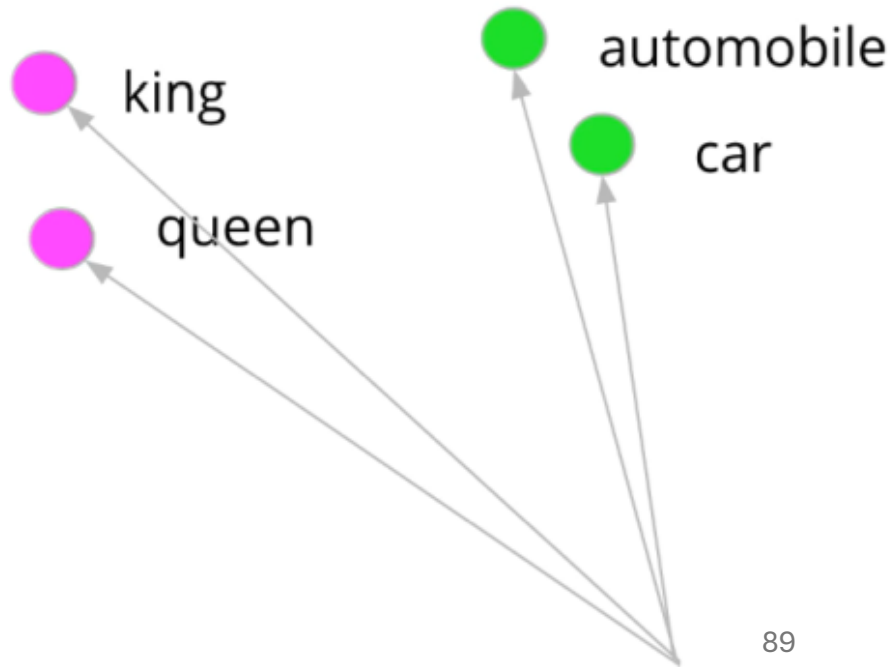
Credit: AbsolutelyAwesome (@Youtube)

Tensorflow Embedding

- This is exactly what the Embedding layer does!
- STEP #1: Convert words into integers
  - [“I”, “Like”, “Cats”] → [50, 25, 3]
- STEP #2: Use integers to index the word embedding matrix to get word vectors for each word
  - [50, 25, 3] → [[0.3, -0.5], [1.2, -0.7], [-2.1, 0.9]]
  - T-length array → T x D matrix

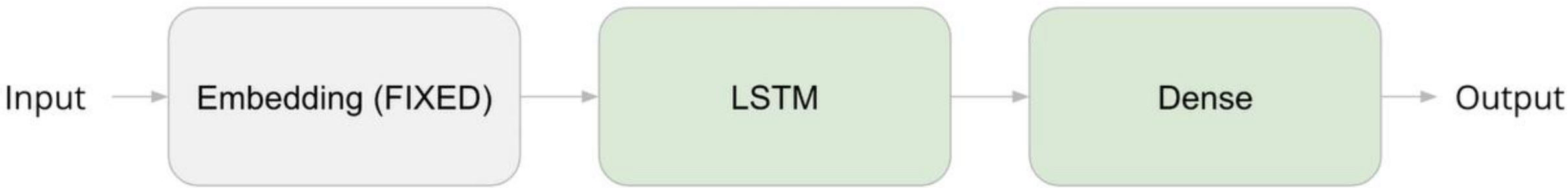
How are the weights found?

- We know intuitively that the word vectors must have some useful structure
- Luckily, it's the same story as with convolutional filters
- These weights are found *automatically* with gradient descent when you call `model.fit()`



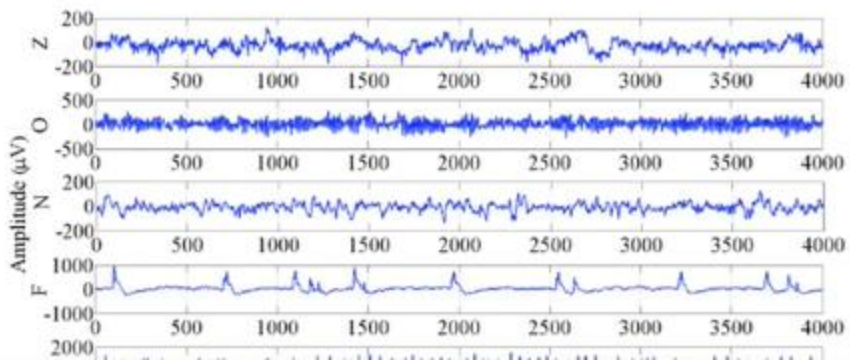
Pre-trained word vectors

- Sometimes, we use *pre-trained* word vectors (trained using some other algorithm)
- We freeze (fix) the embedding layer's weights, so only the *other* layers are trained when we call `model.fit()`
- You're encouraged to read about word2vec and GloVe to learn more



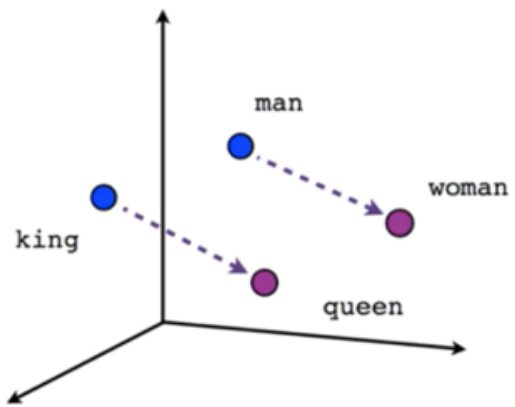
Summary

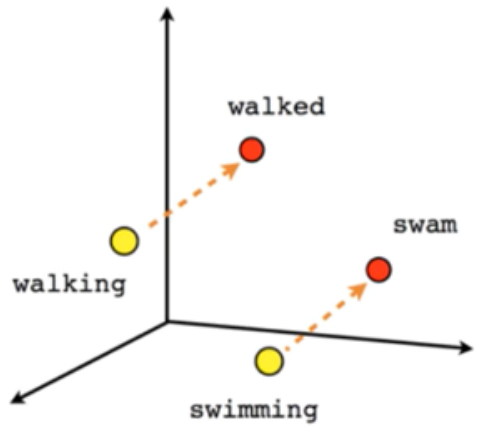
- It's not possible to multiply a *word* by a weight matrix - there is no concept of multiplying words
- We must turn words into some kind of numerical representation first
- One-hot encoding is a fine first guess
  - Too much space (there could be up to 1 million tokens)
  - Not geometrically useful ("cat" is just as close to "feline" as it is to "airplane")
  - We need to make use of "machine learning is nothing but a geometry problem"
- Better: convert words to unique integers
- Use those integers to *index* a weight matrix (the *embedding matrix*)
- Much faster than matrix multiplication

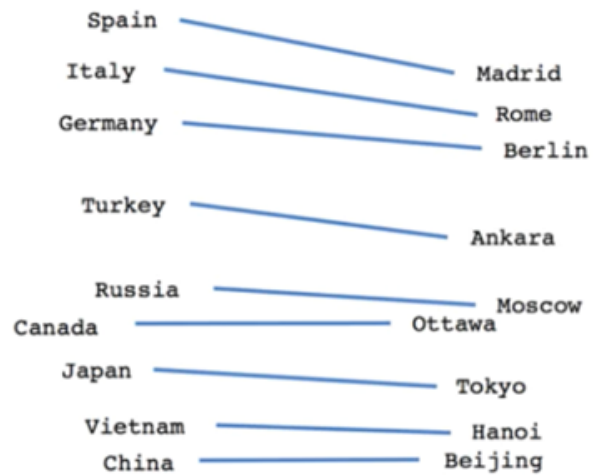


Summary

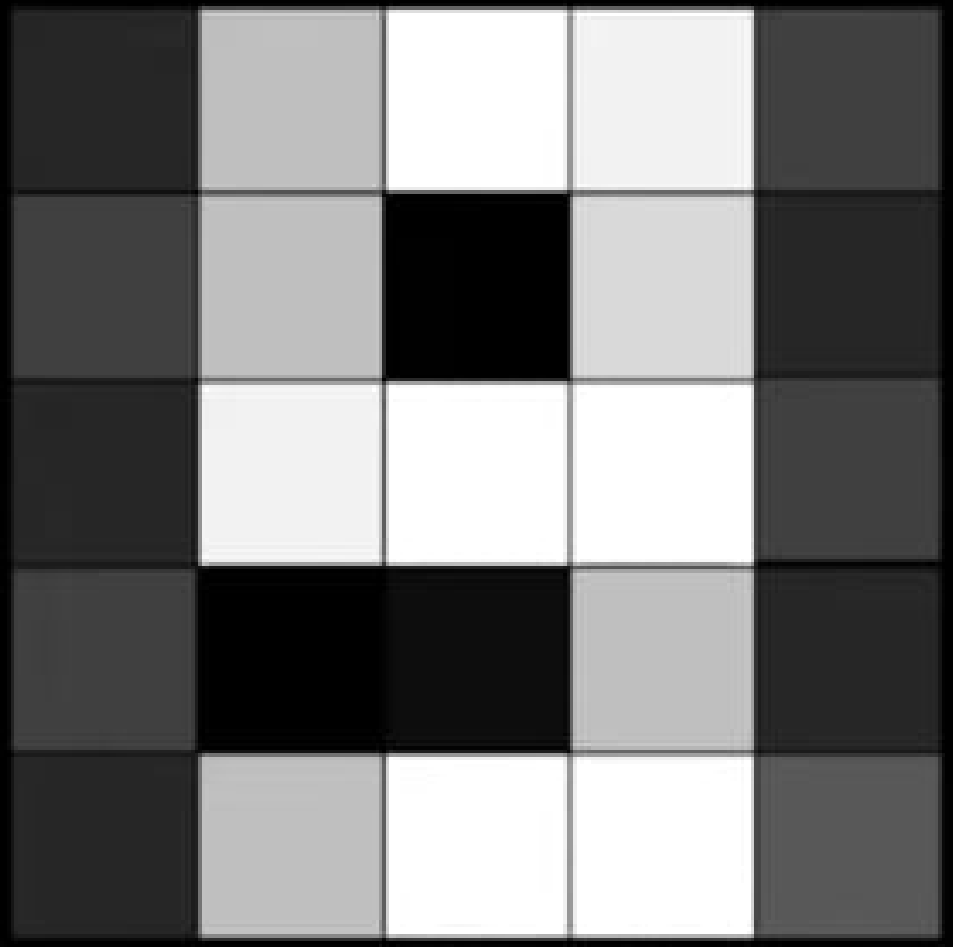
- Each input sentence thus becomes a T x D sequence which we know an RNN can accept
- The embedding is trained like any other layer - we just need to call `model.fit()`
- Read up on word2vec and GloVe for alternative methods for training embeddings







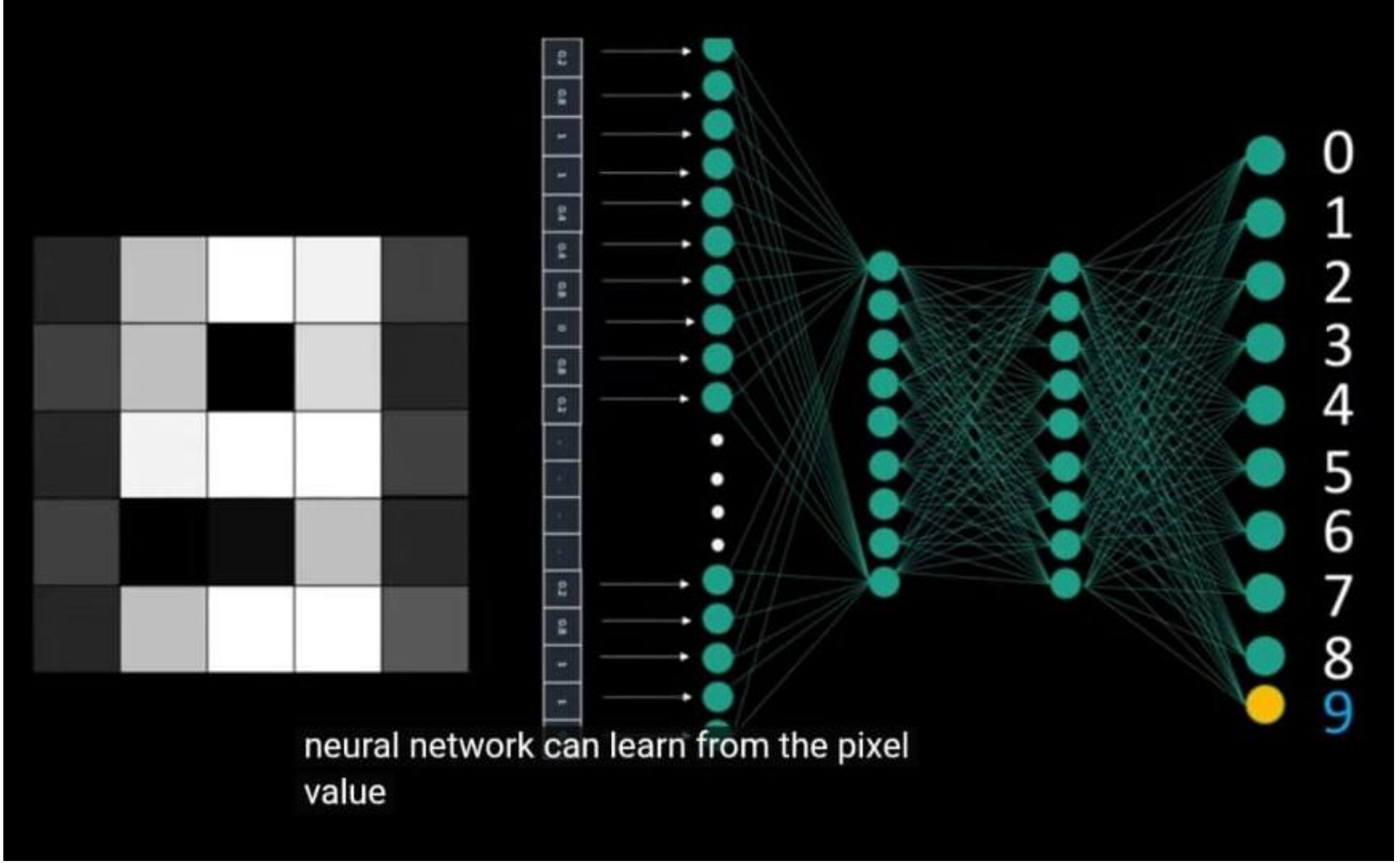
GRAYSCALE IMAGE OF DIGIT 9



NUMERICAL REPRESENTATION

0.2	0.8	1	1	0.4
0.4	0.8	0	0.8	0.2
0.2	0.9	1	1	0.4
0.8	0	0.3	0.8	0.3
0.2	0.8	1	1	0.4

Credit: AbsolutelyAwesome (@Youtube)



Credit: AbsolutelyAwesome (@Youtube)

ONE HOT ENCODING

OHE Avoids Numeric Relations

Simple to use

It is Sparse

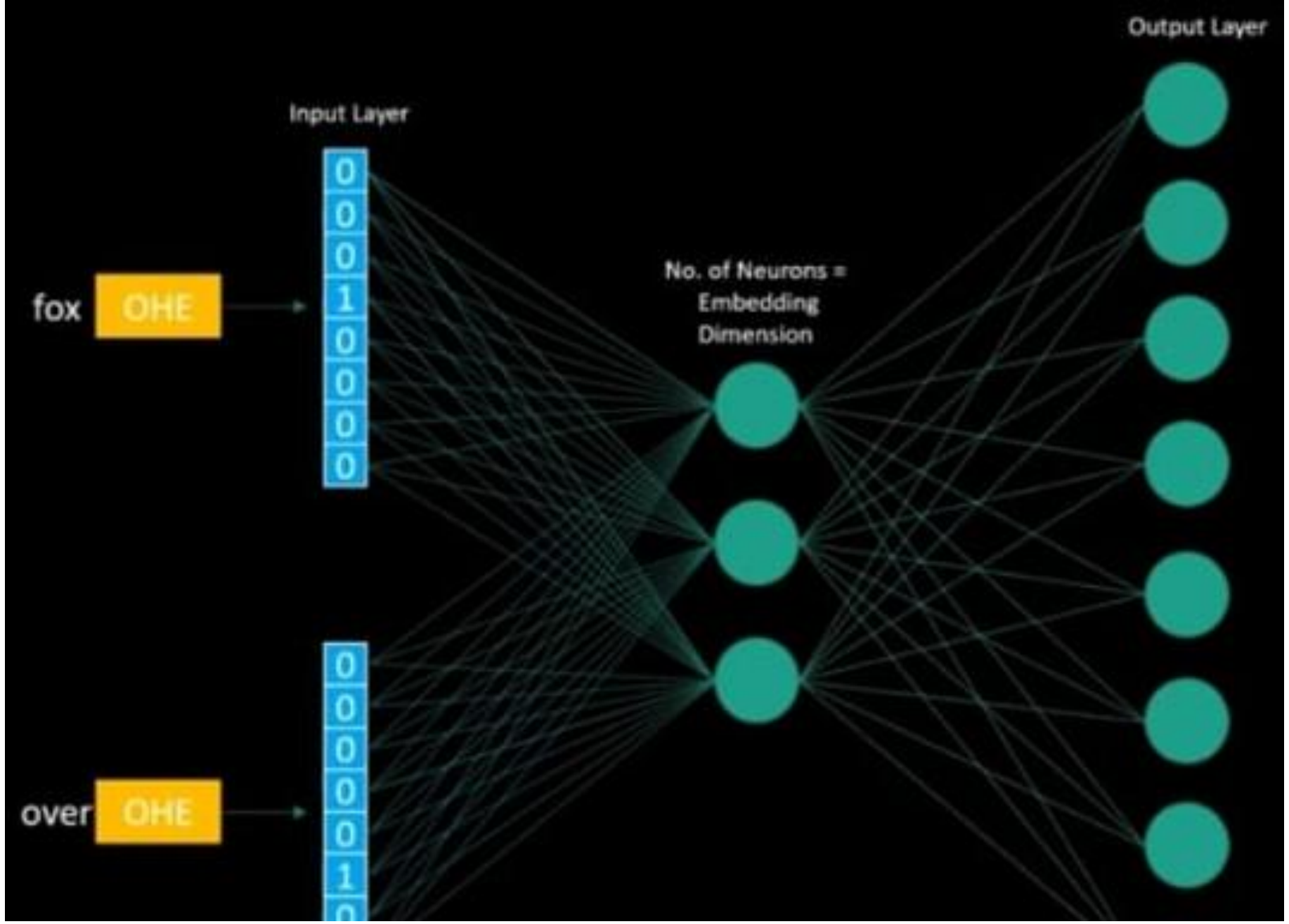
n-size vector for n-vocab size

Cannot capture semantic meaning and contextual understanding

For GPT-2 50257 vocab
Large Sparse Matrix of 50257 Dimension

[0 0 0 0 1 . . . . . 0]

Credit: AbsolutelyAwesome (@Youtube)



Credit: AbsolutelyAwesome (@Youtube)

Sentence 1: This movie was good
Sentence 2: This movie was bad

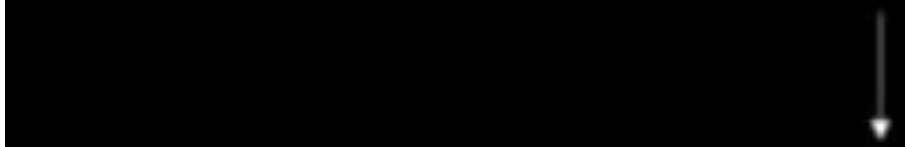
Vocabulary: ["This", "movie", "was", "good", "bad"]
Vocabulary Size: 5

ONE HOT ENCODING TABLE

	This	movie	was	good	bad
This	1	0	0	0	0
movie	0	1	0	0	0
was	0	0	1	0	0
good	0	0	0	1	0
bad	0	0	0	0	1

Credit: AbsolutelyAwesome (@Youtube)

Sentence 1: This movie was good
Sentence 2: This movie was bad



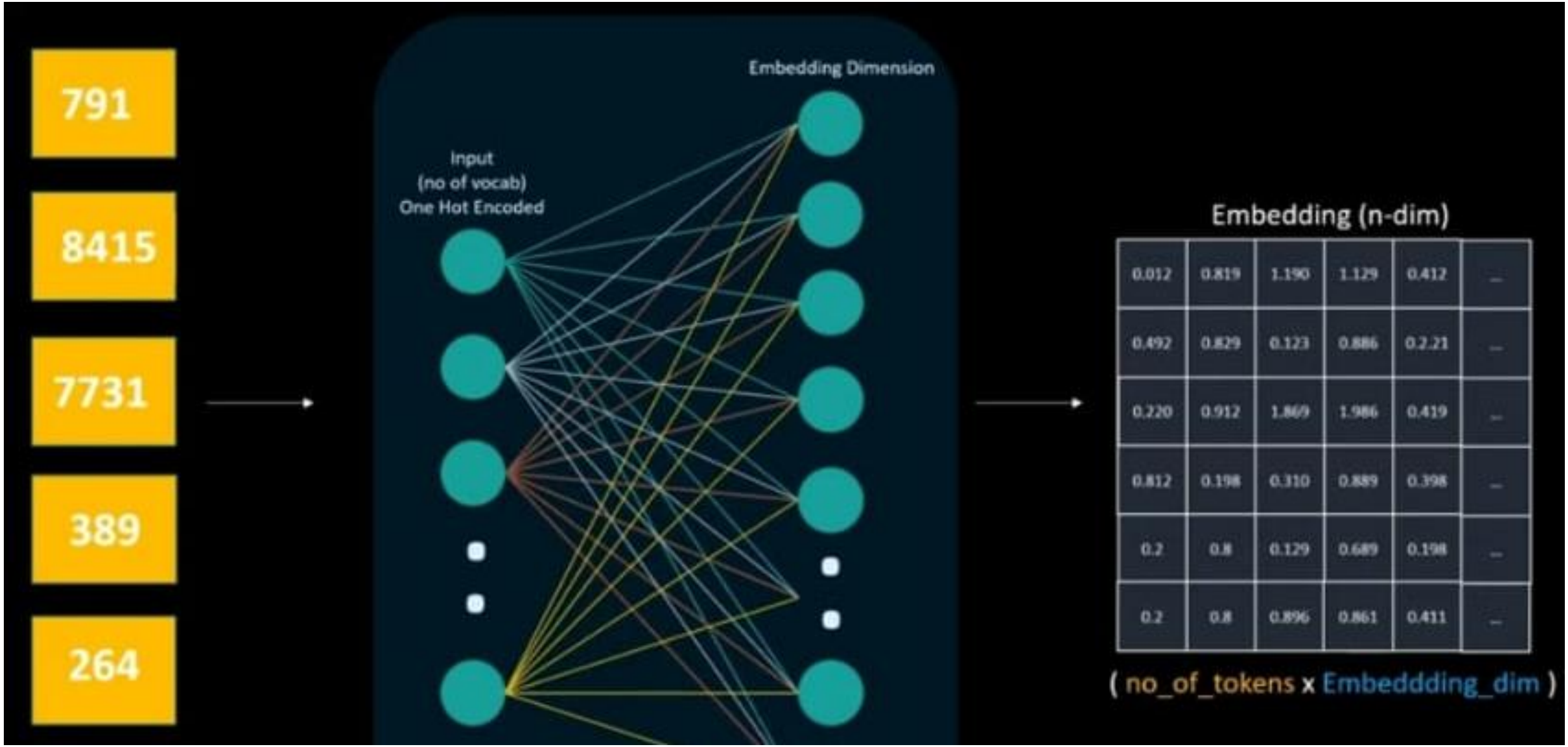
	This movie	movie was	was good	was bad
Sentence 1	1	1	1	0
Sentence 2	1	1	0	1

Bi-Gram (2 word context)

N-Gram with N-Word Context

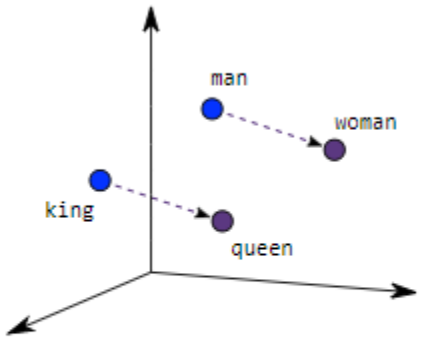
Sparsity problem in large corpora where there are many possible n-gram

Credit: AbsolutelyAwesome (@Youtube)

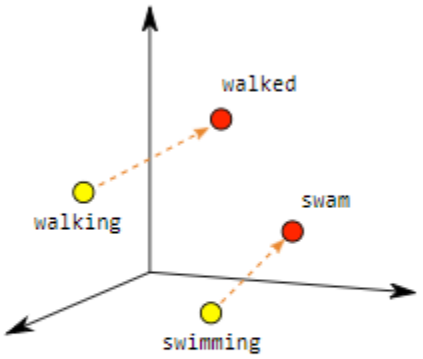


Credit: AbsolutelyAwesome (@Youtube)

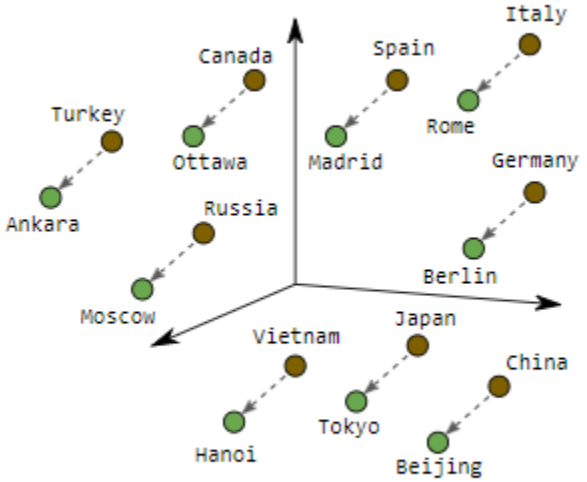
Word Embeddings



Male-Female



Verb Tense

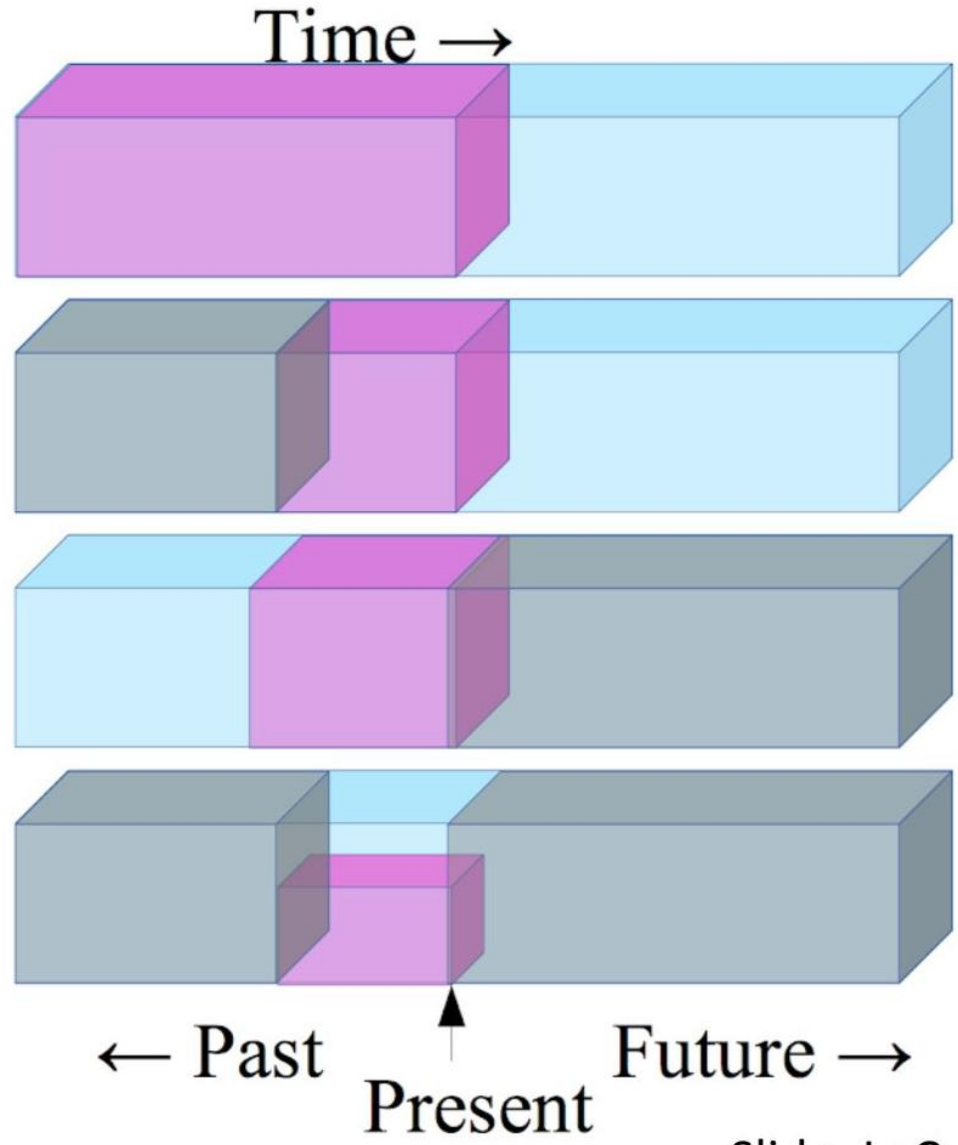


Country-Capital

Credit to: Lazy Programmer, Ariel Bar-itzhack

Predicting neighbouring context

- **Predict any part of the input from any other part.**
- **Predict the future from the past.**
- **Predict the future from the recent past.**
- **Predict the past from the present.**
- **Predict the top from the bottom.**
- **Predict the occluded from the visible**
- **Pretend there is a part of the input you don't know and predict that.**

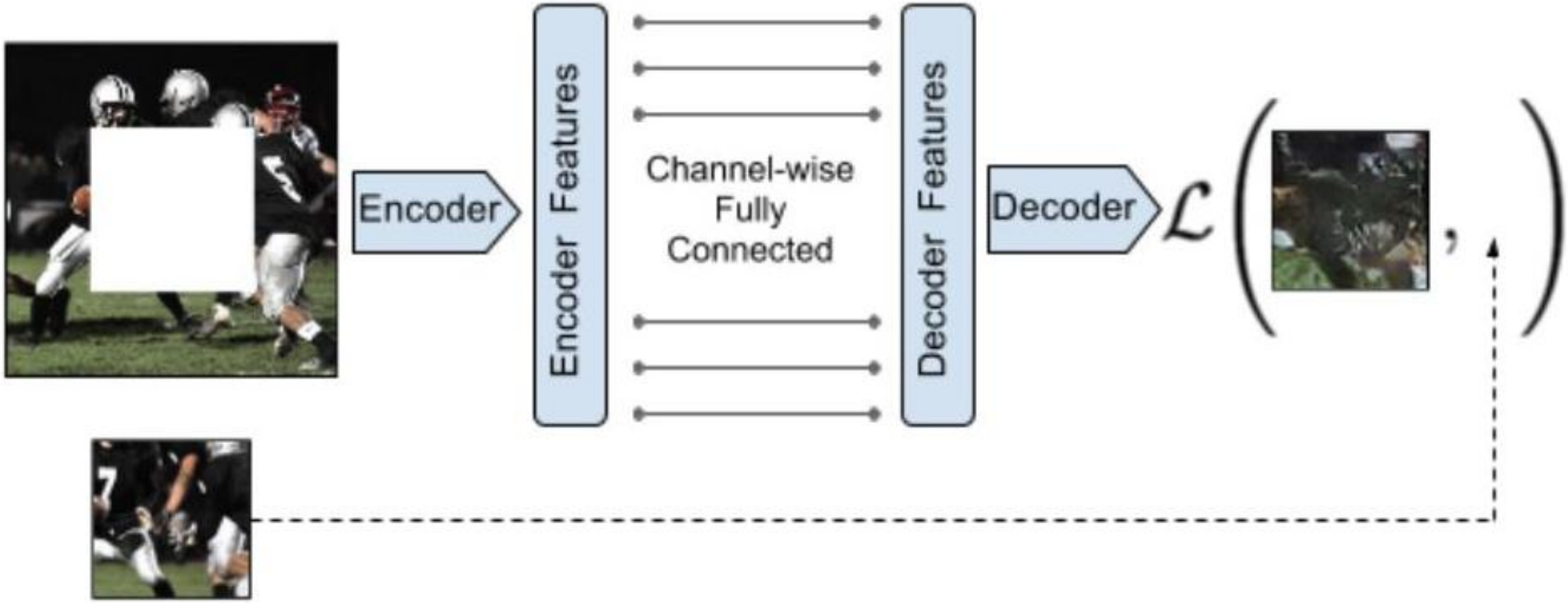


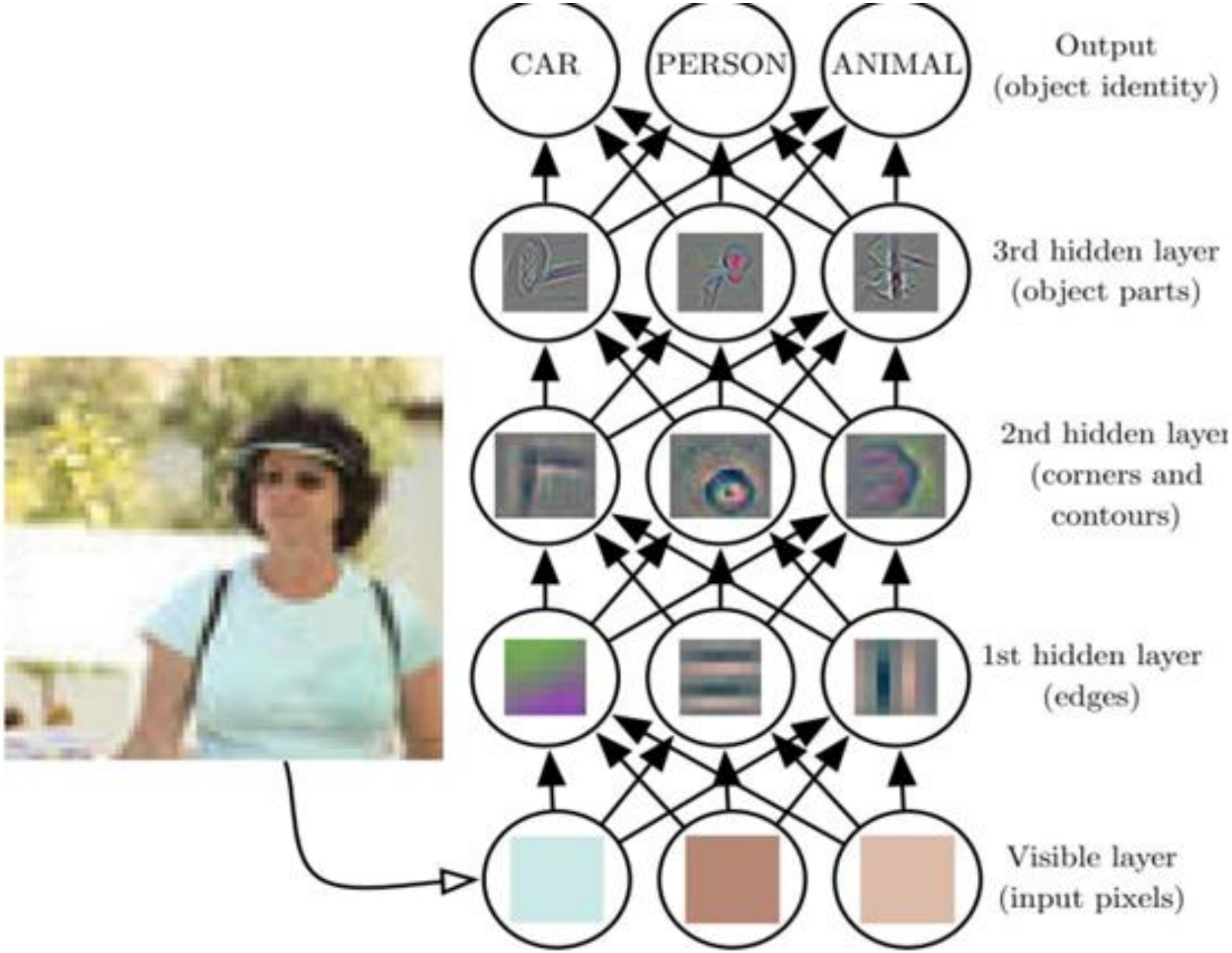
Slide: LeCun

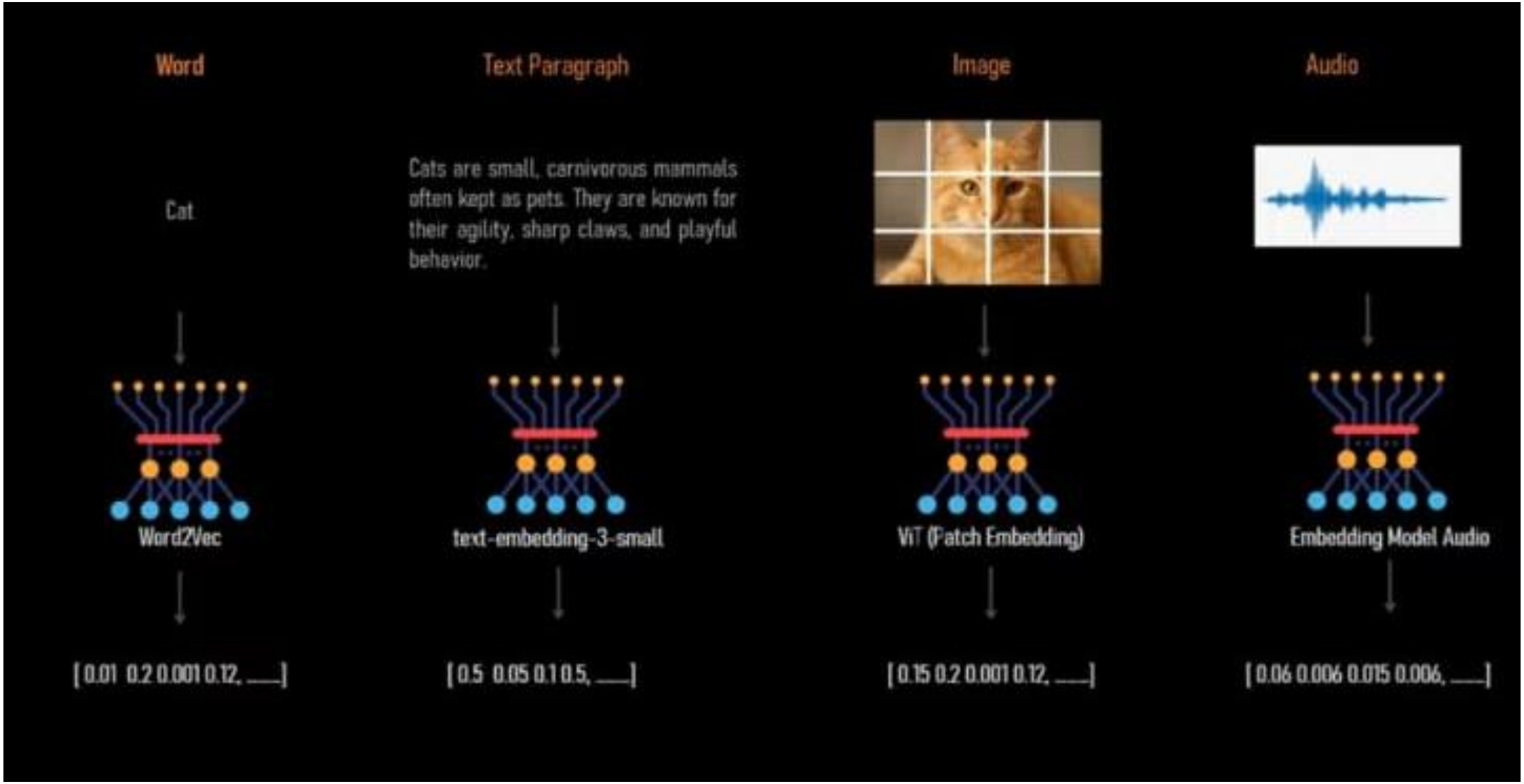
- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

Predict missing pieces

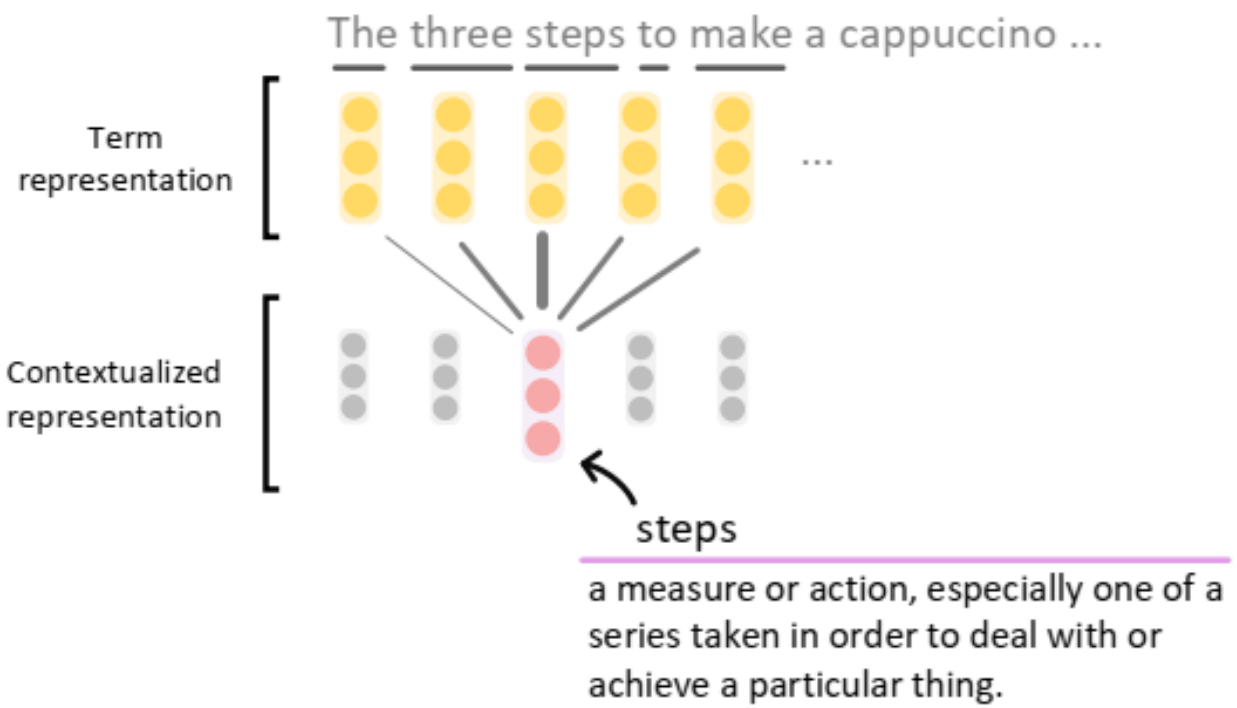
Context Encoders



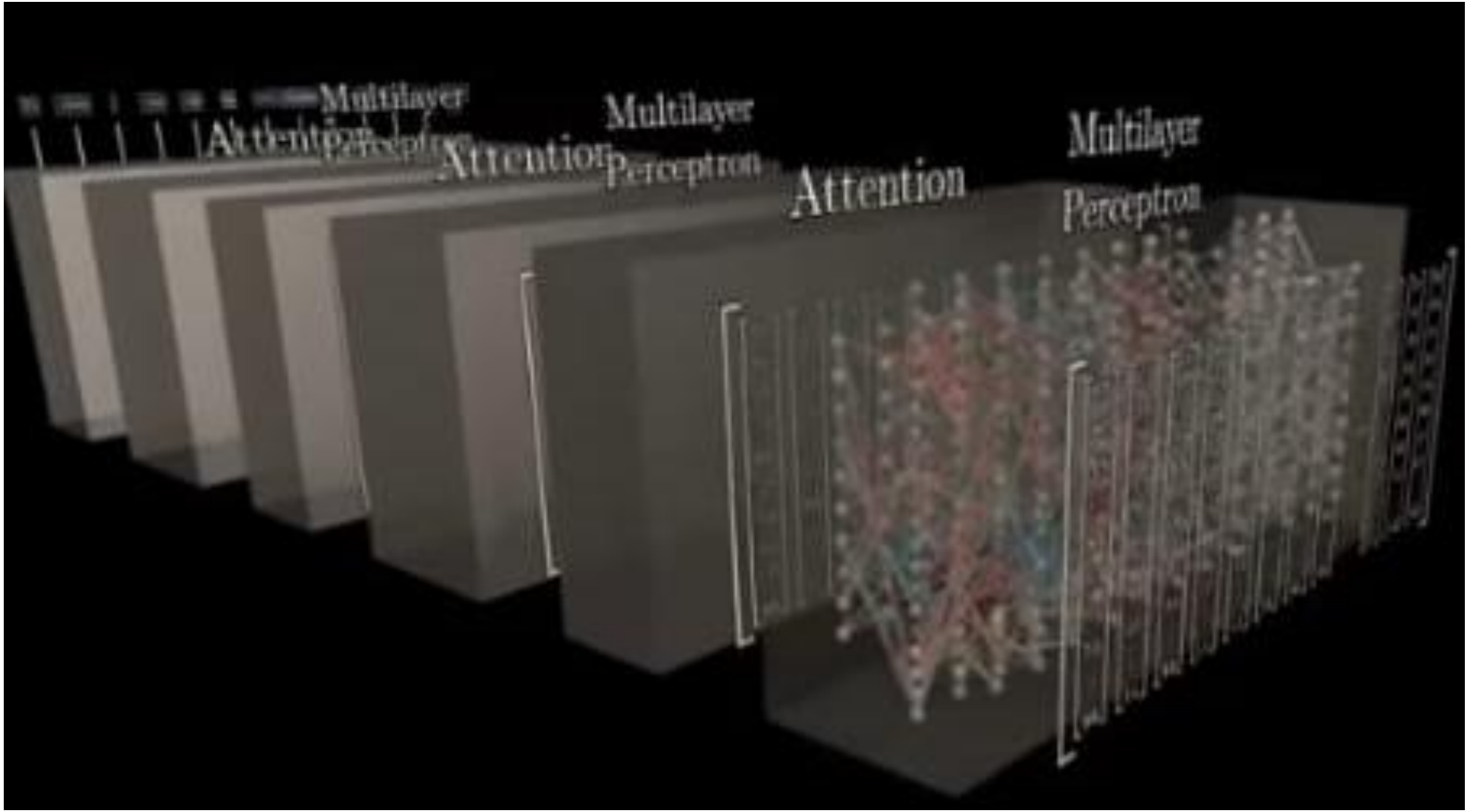

CAR
PERSON
ANIMAL
Output
(object identity)
3rd hidden layer
(object parts)
2nd hidden layer
(corners and
contours)
1st hidden layer
(edges)
Visible layer
(input pixels)


Word
Cat
Word2Vec
[0.01 0.2 0.001 0.12, ____]
Text Paragraph
Cats are small, carnivorous mammals often kept as pets. They are known for their agility, sharp claws, and playful behavior.
text-embedding-3-small
[0.5 0.05 0.1 0.5, ____]
Image
ViT (Patch Embedding)
[0.15 0.2 0.001 0.12, ____]
Audio
Embedding Model Audio
[0.06 0.006 0.015 0.006, ____]

Contextualization via Self-Attention



- Learn meaning based on surrounding context for every word occurrence
- This *contextualization* combines representations
- Context here is local to the sequence (not necessary a fixed window)
- Is computationally intensive $O(n^2)$
  - Every token attends to every other token

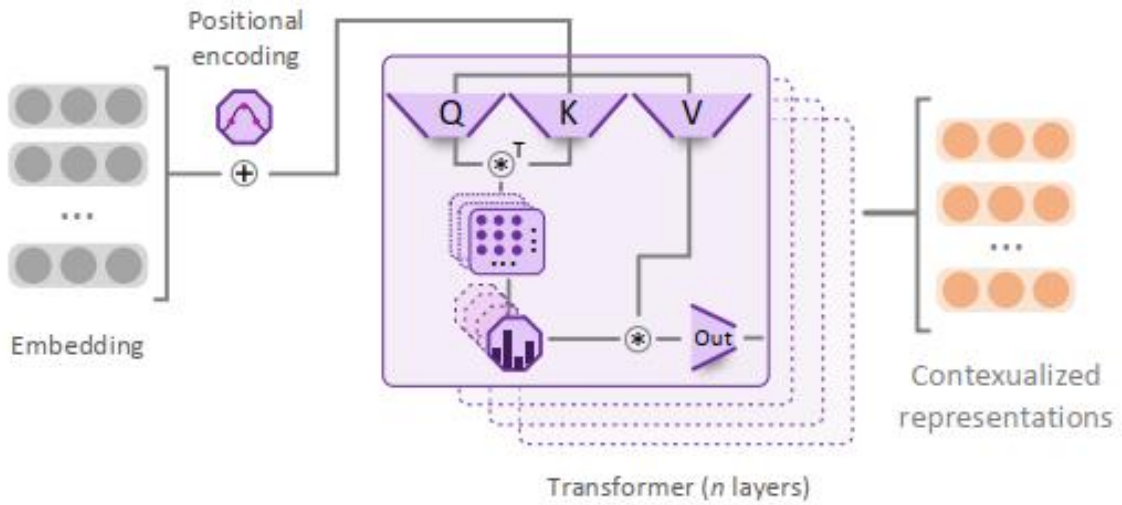

Multilayer
Perceptron
Attention
Multilayer
Perceptron
Attention
Multilayer
Perceptron

Transformer

- Transformers contextualize with multi-head self-attention
  - Every token attends to every other token $O(n^2)$ complexity
- Commonly Transformers stack many layers
- Can be utilized as encoder-only or encoder-decoder combination
- Do not require any recurrence
  - The attention breaks down to a series of matrix multiplications over the sequence
- Initially proposed in translation
  - Now the backbone of virtually every NLP advancement in the last years

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al.
Attention is all you need. In NeurIPS. 2017.

Transformer – Architecture



- We embed (subword) tokens
- We add a positional encoding
- In each Transformer-Layer:
  - Project each vector with 3 linear layers to **Q**uery, **K**ey, **V**alue
  - Transform projections to another multi-head dimension
  - Matrix-multiply Query & Key
  - Get Q-K attention via softmax
  - Multiply attention with Values and project back to output

Nice detailed walkthrough code + paper:
https://nlp.seas.harvard.edu/2018/04/03/attention.html

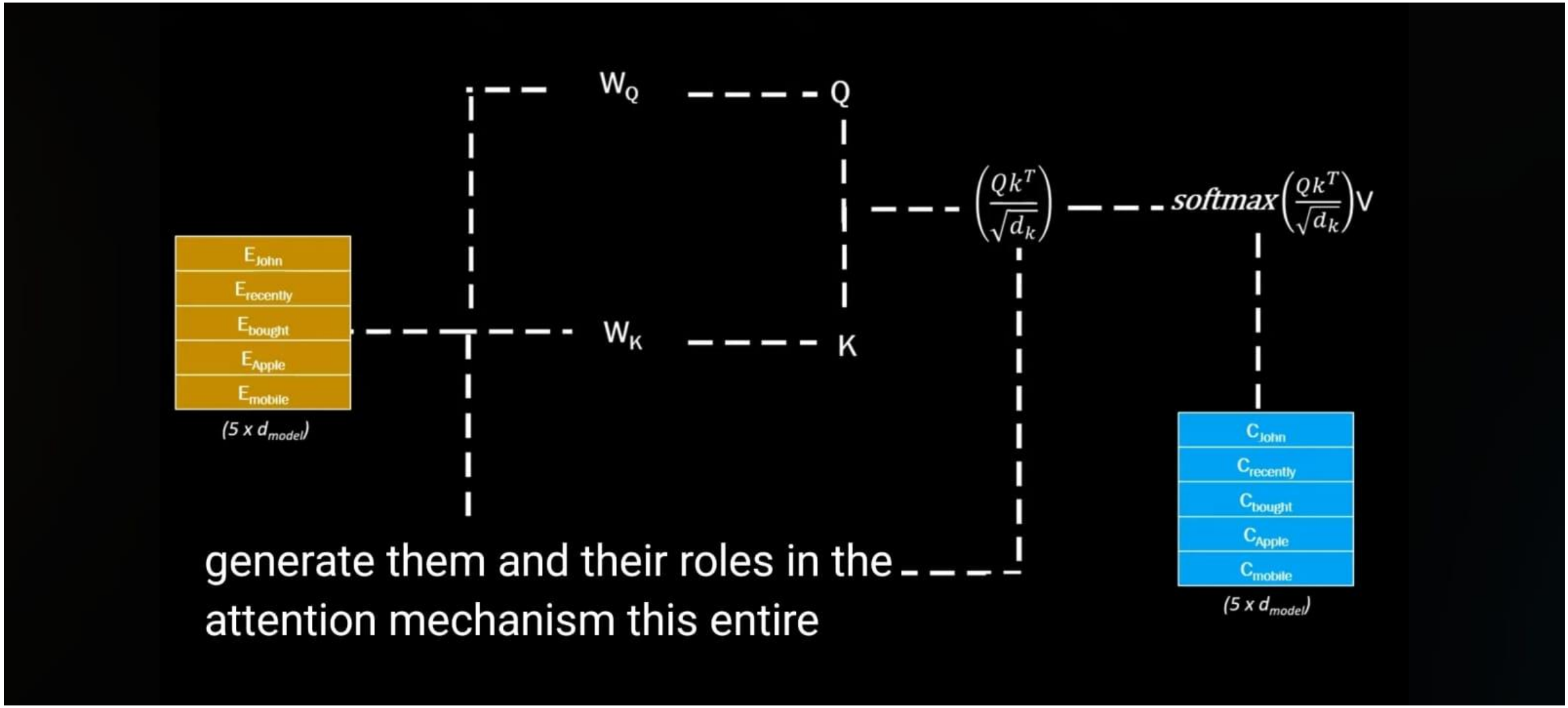
Self-Attention Definition

- The Transformer Self-Attention is defined as:

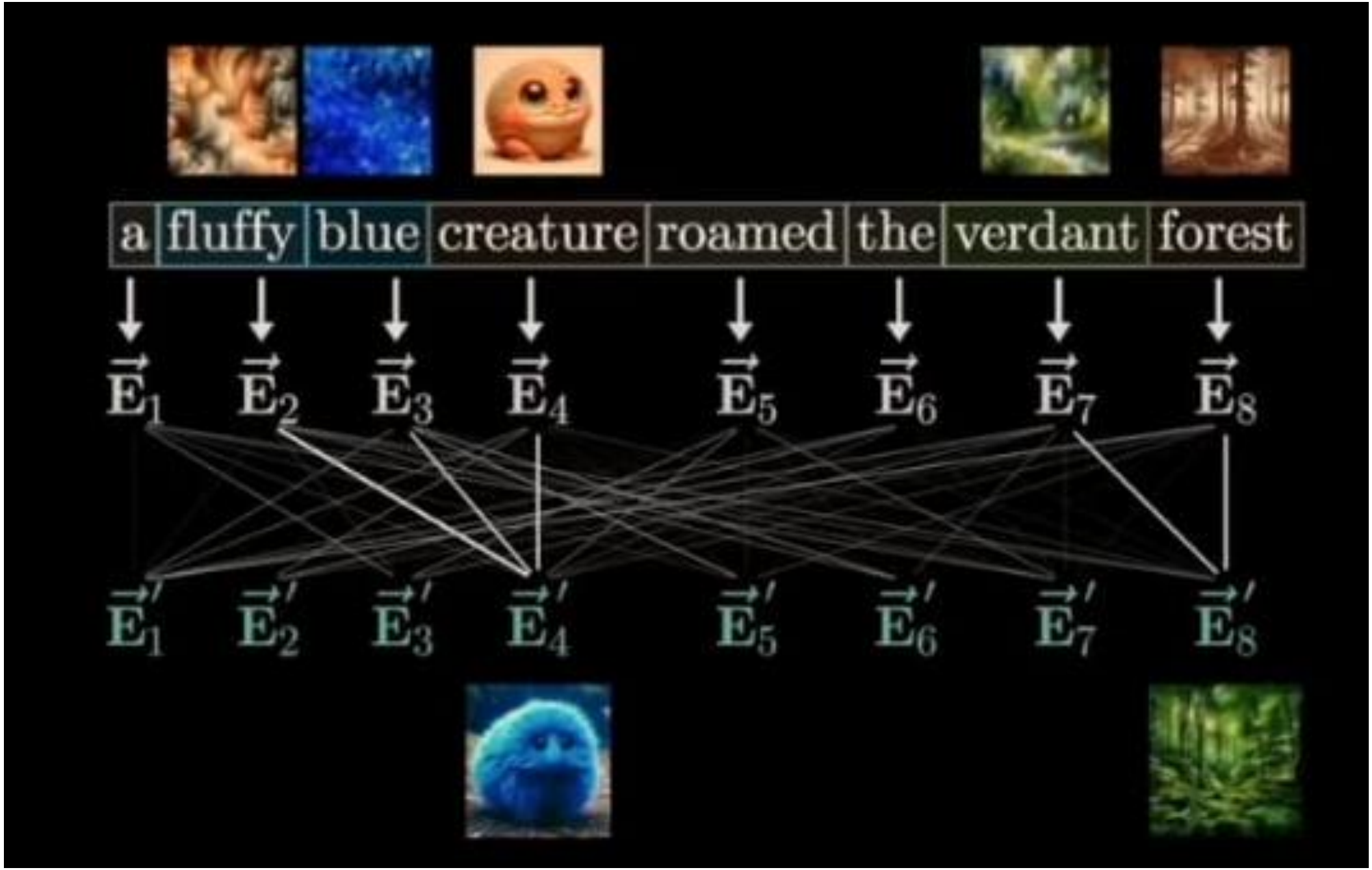
$$\text{SelfAttention}(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) * V$$

- Q, K, V are projections of the **same input** sequence
- This definition hides quite a bit of complexity, visible in the code

Q	Attention "Query"
K	Attention "Key"
V	Attention "Value"
d_k	Dimension of key embeddings

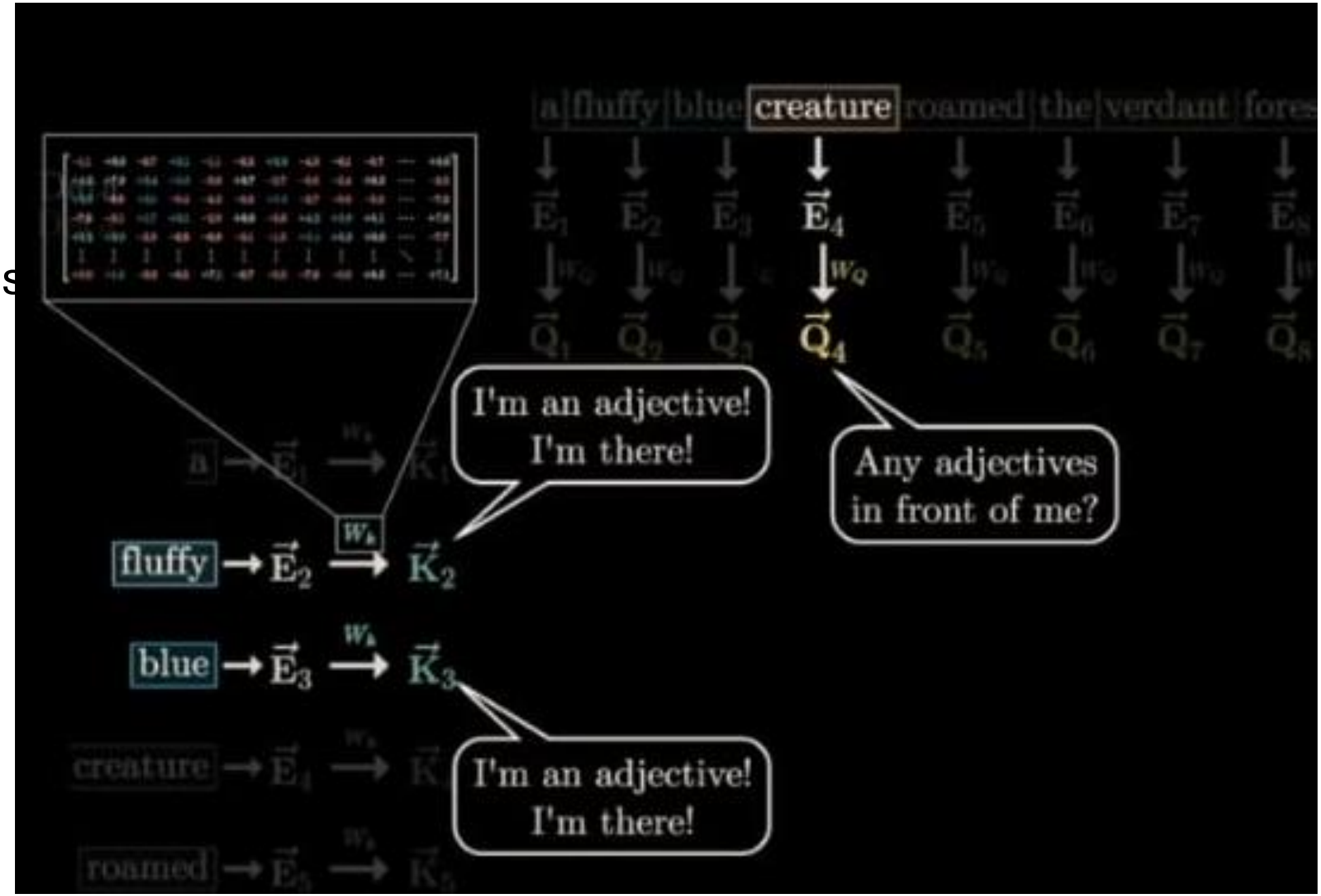


Credit: Grant Sanderson

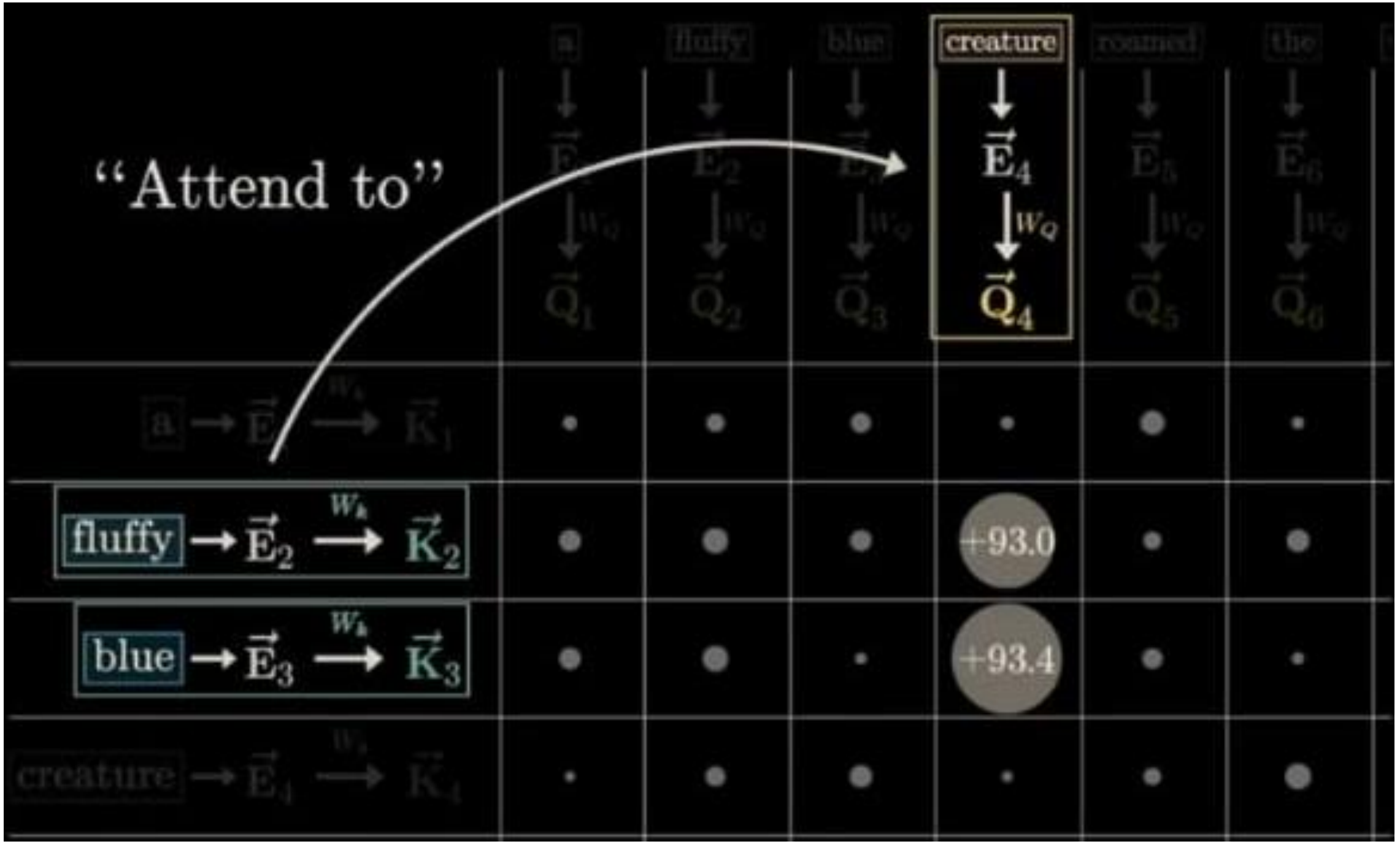


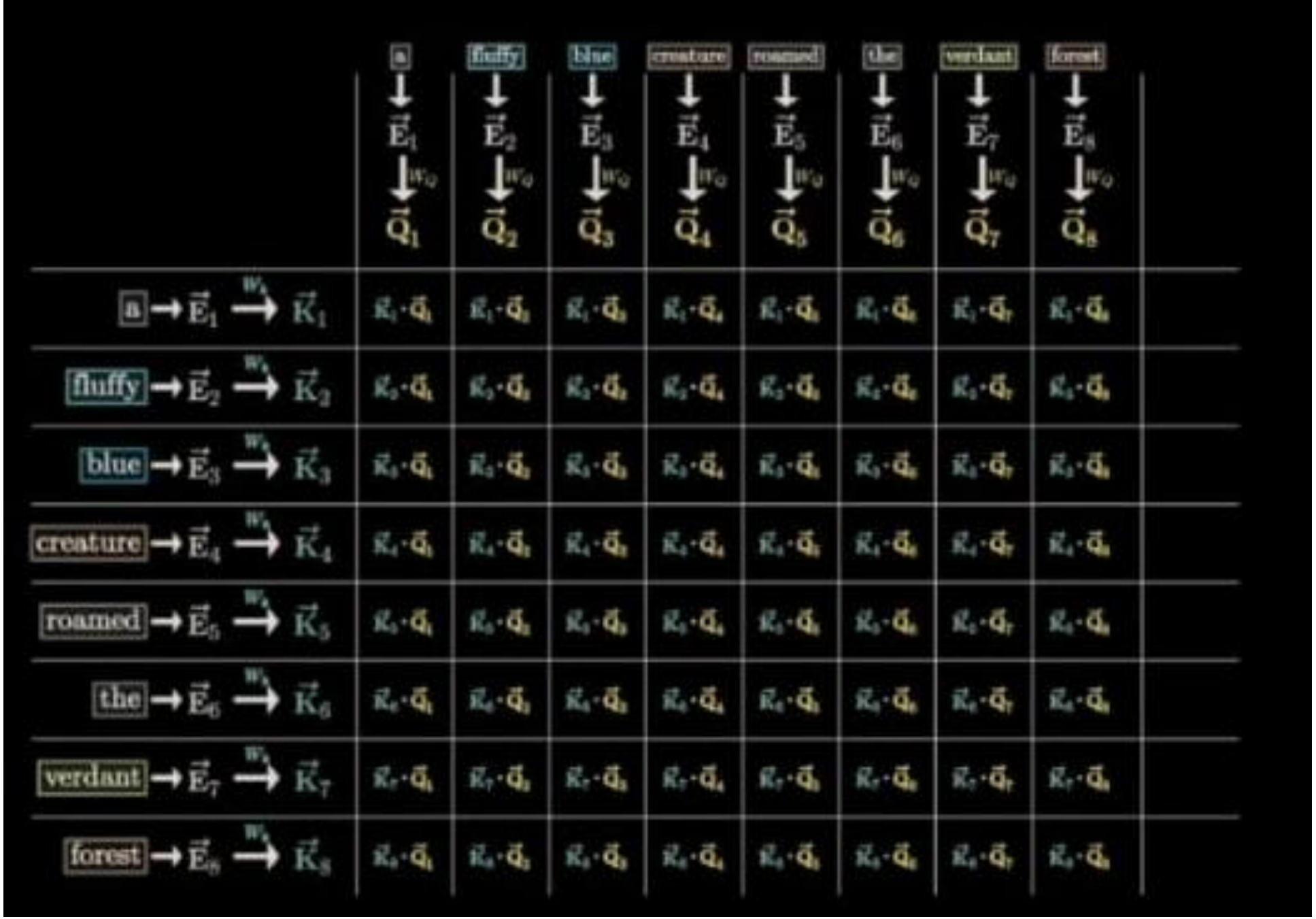
Credit: Grant Sanderson

- S

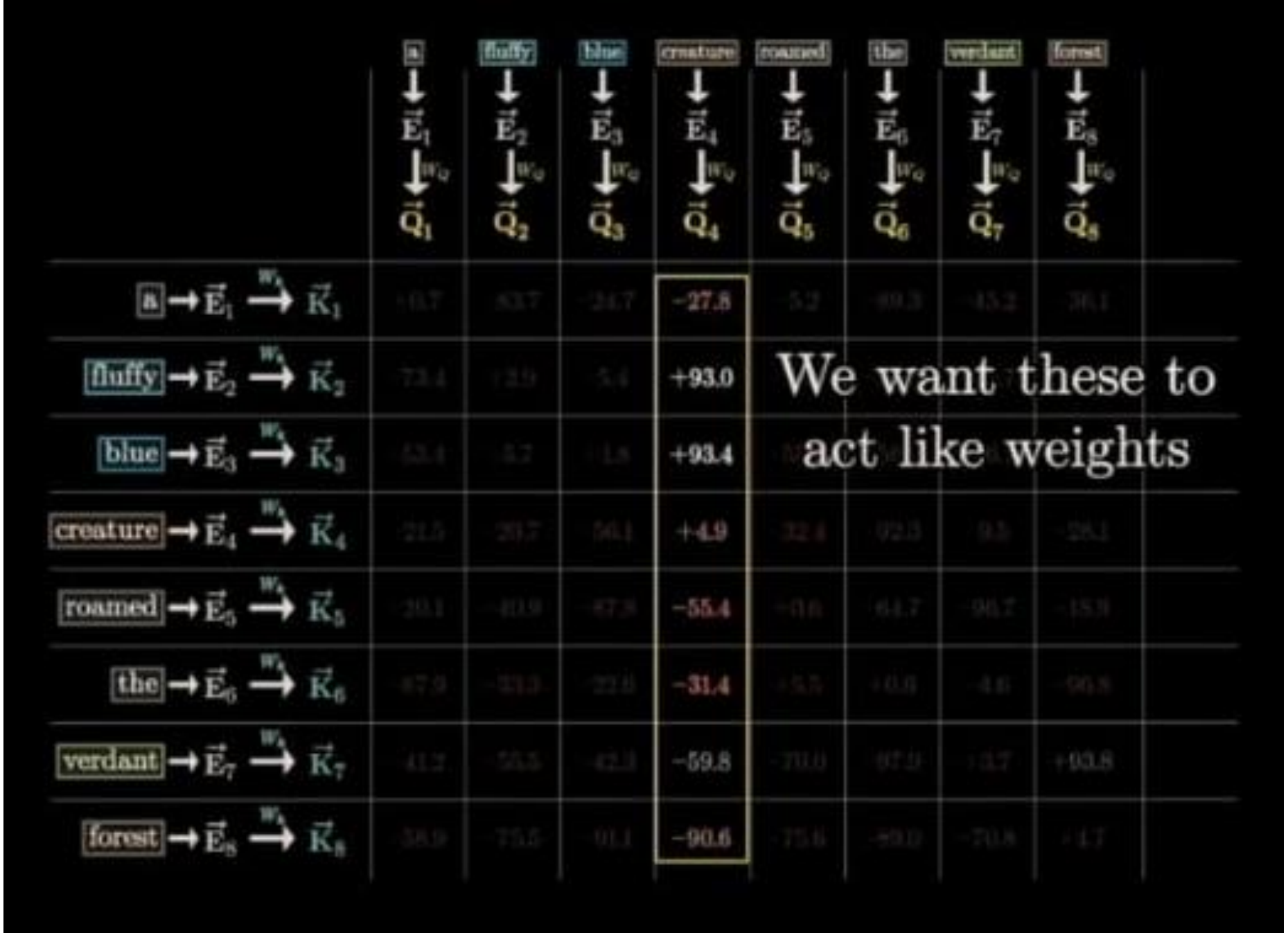


Credit: Grant Sanderson

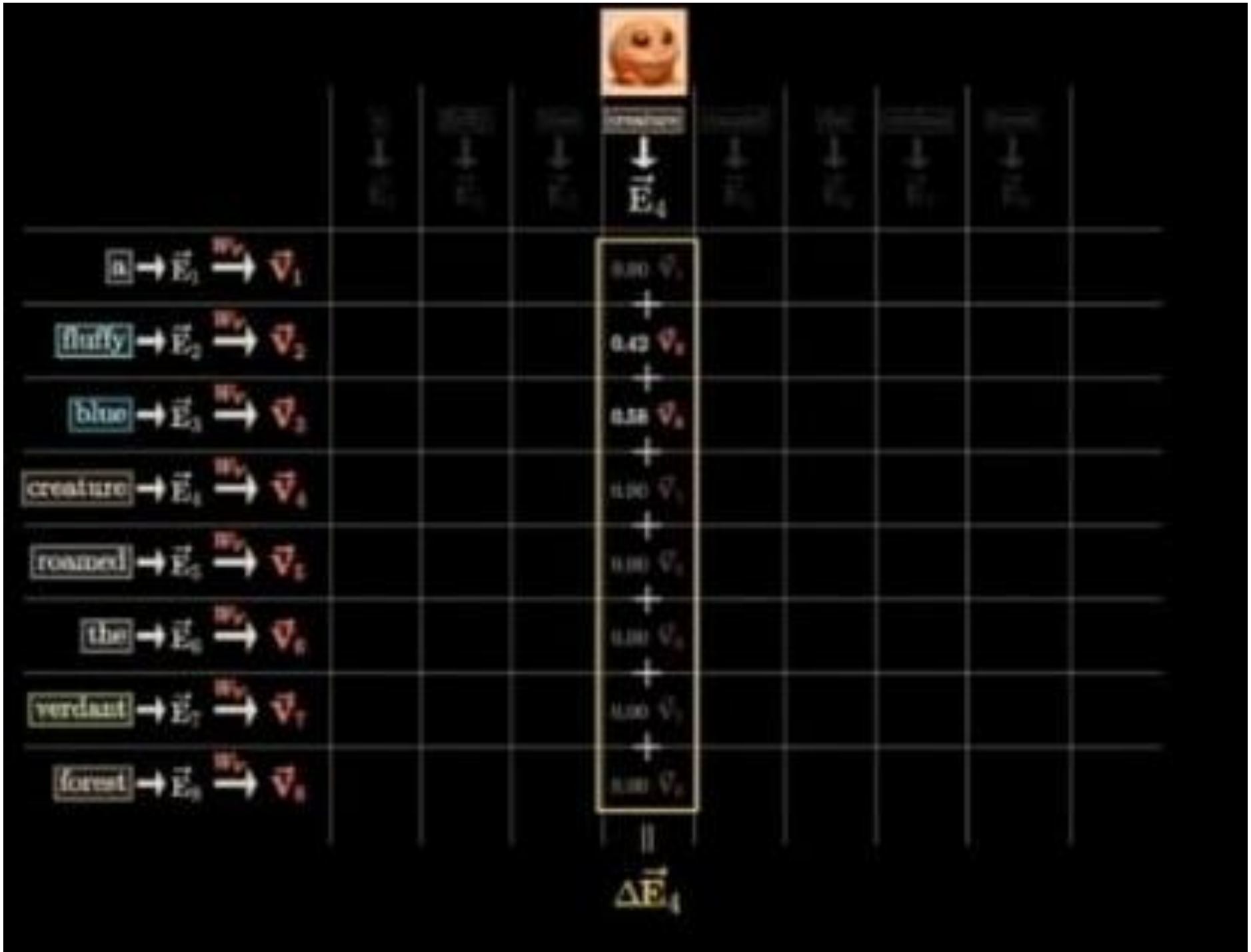

"Attend to"
a
fluffy
blue
creature
roamed
the
$\vec{E}_1$
$\vec{E}_2$
$\vec{E}_3$
$\vec{E}_4$
$\vec{E}_5$
$\vec{E}_6$
W_Q
$\vec{Q}_1$
$\vec{Q}_2$
$\vec{Q}_3$
$\vec{Q}_4$
$\vec{Q}_5$
$\vec{Q}_6$
a → $\vec{E}_1$ W_K → $\vec{K}_1$
fluffy → $\vec{E}_2$ W_K → $\vec{K}_2$
blue → $\vec{E}_3$ W_K → $\vec{K}_3$
creature → $\vec{E}_4$ W_K → $\vec{K}_4$
+93.0
+93.4

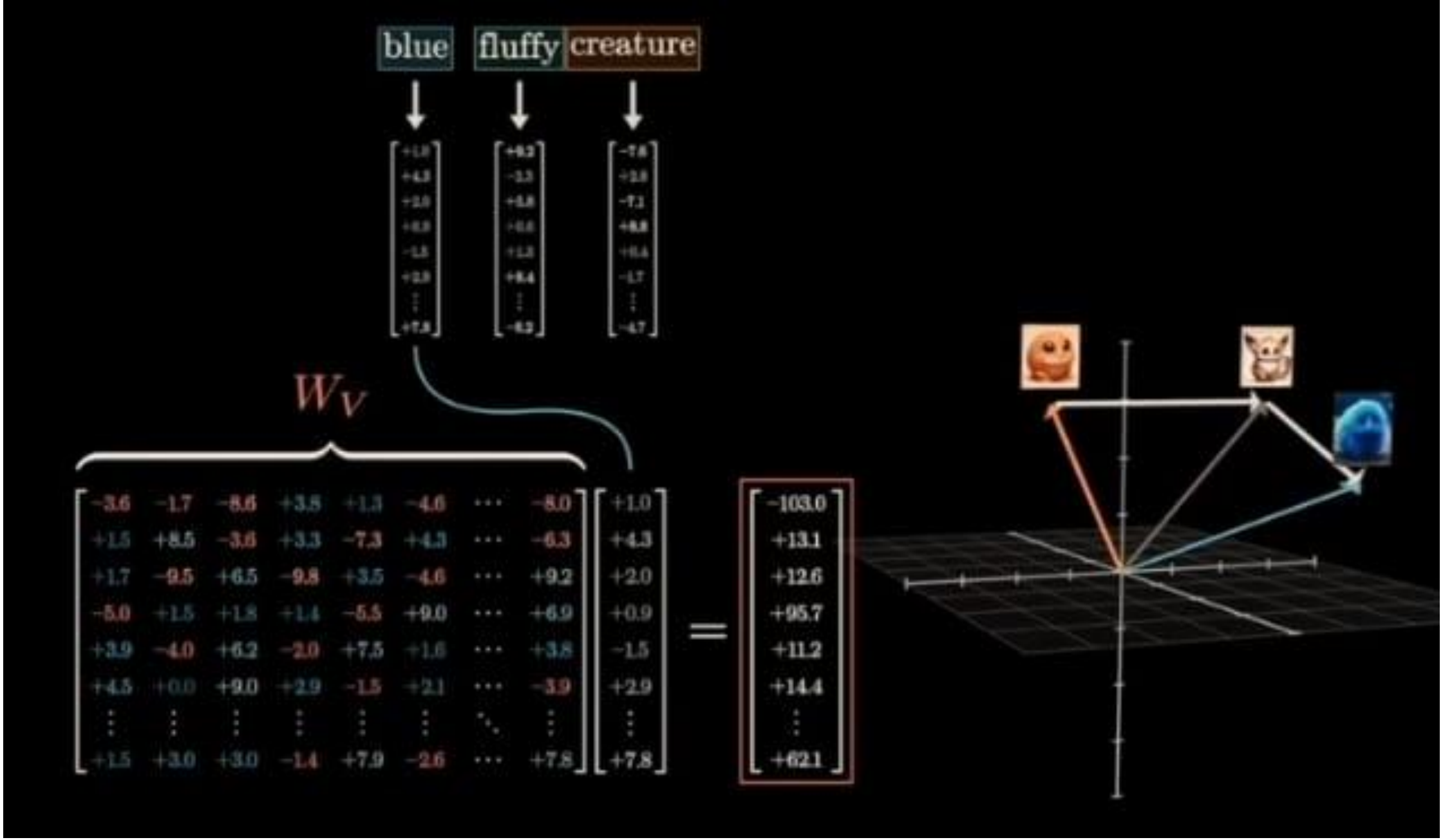


Credit: Grant Sanderson

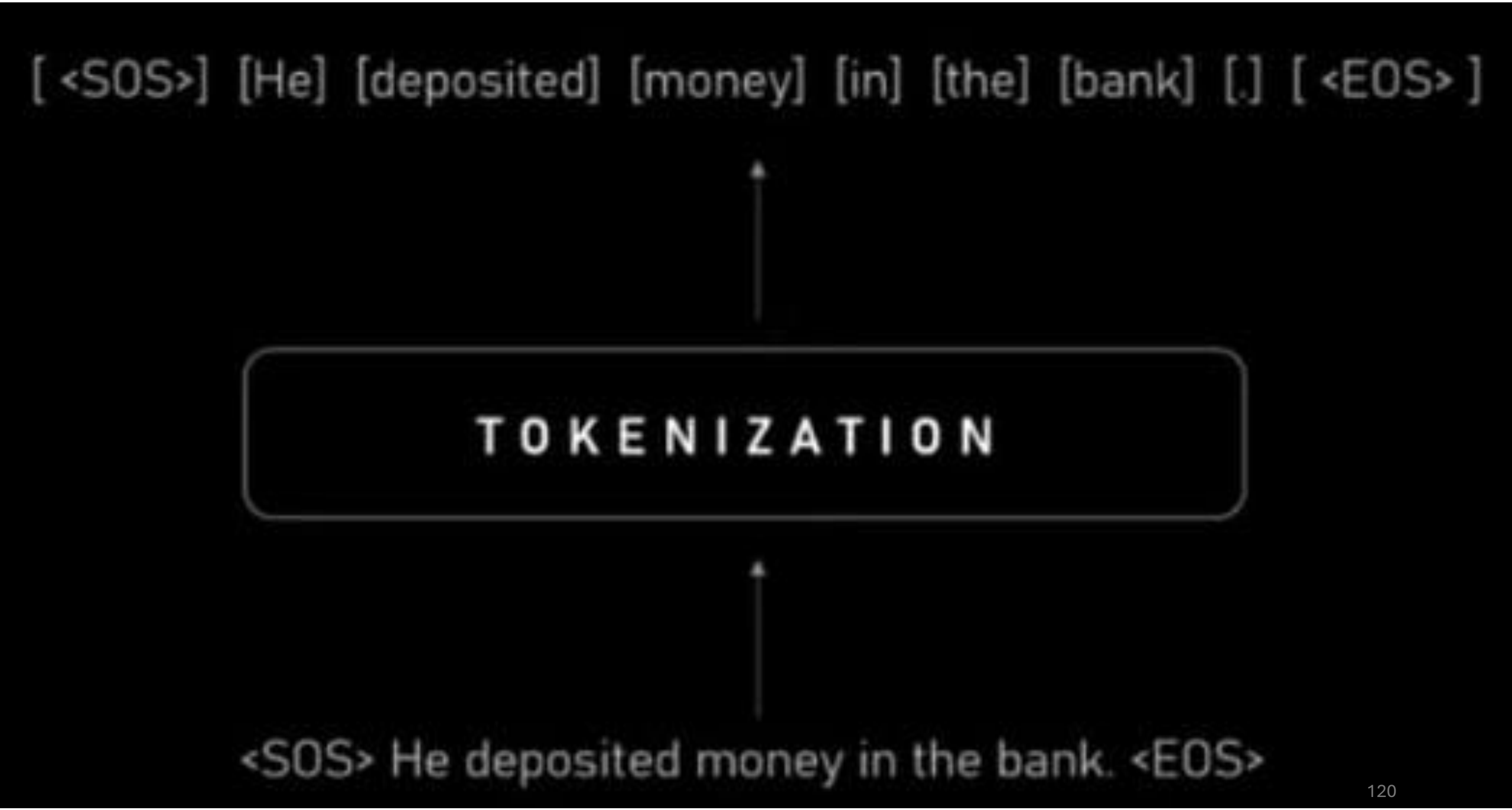


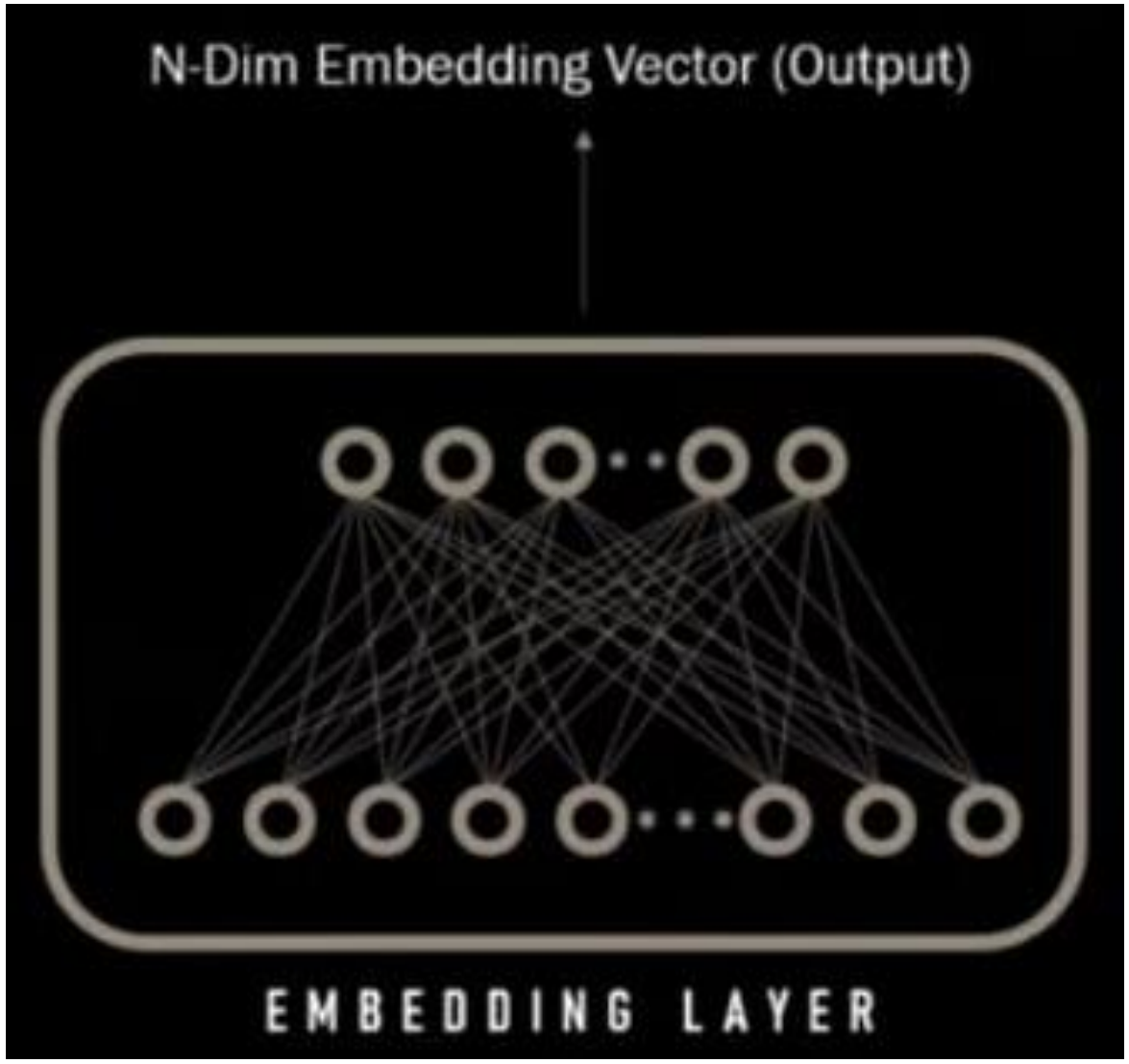
Credit: Grant Sanderson

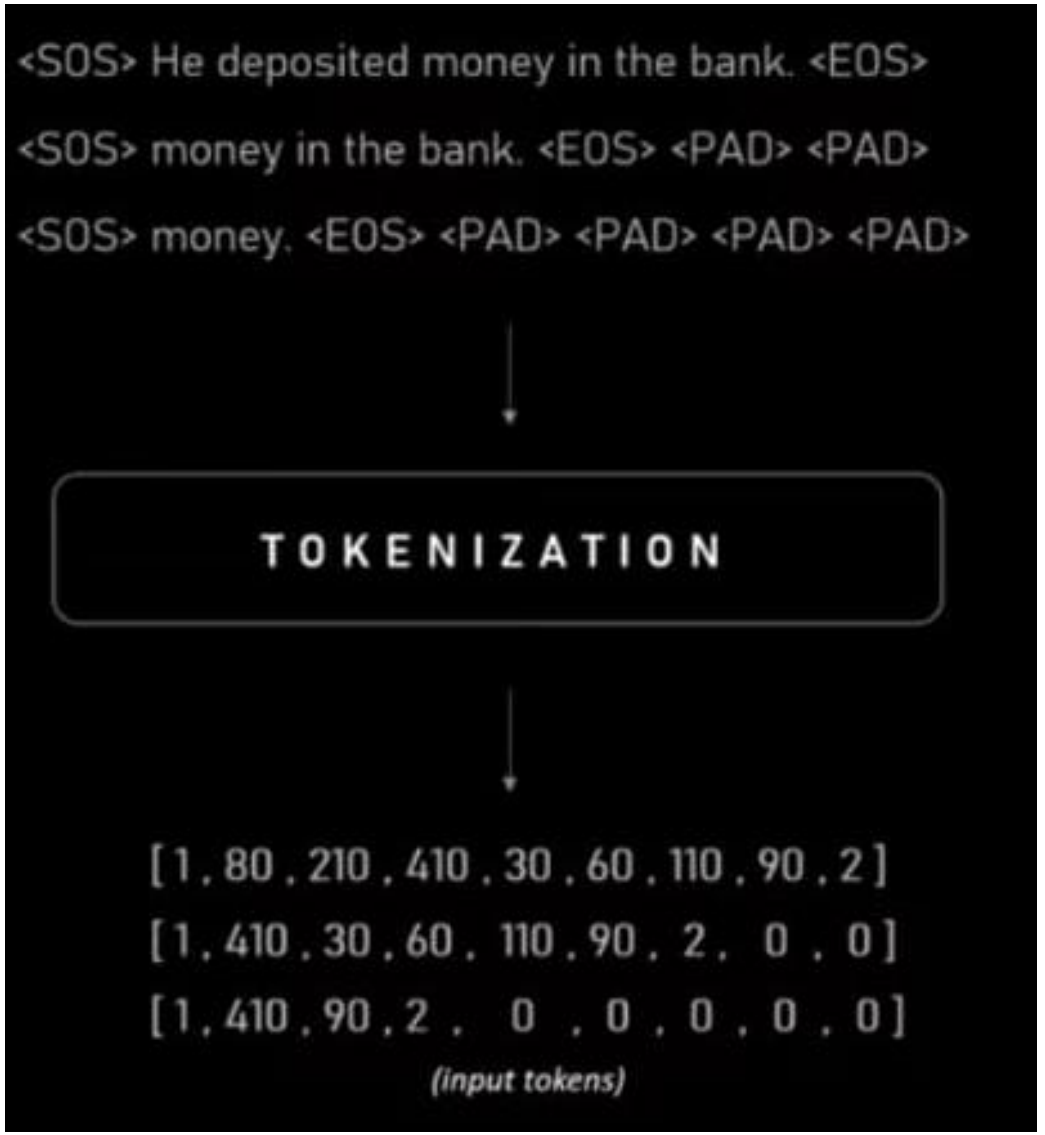

a
fluffy
blue
creature
roamed
the
verdant
forest
0.43
0.58



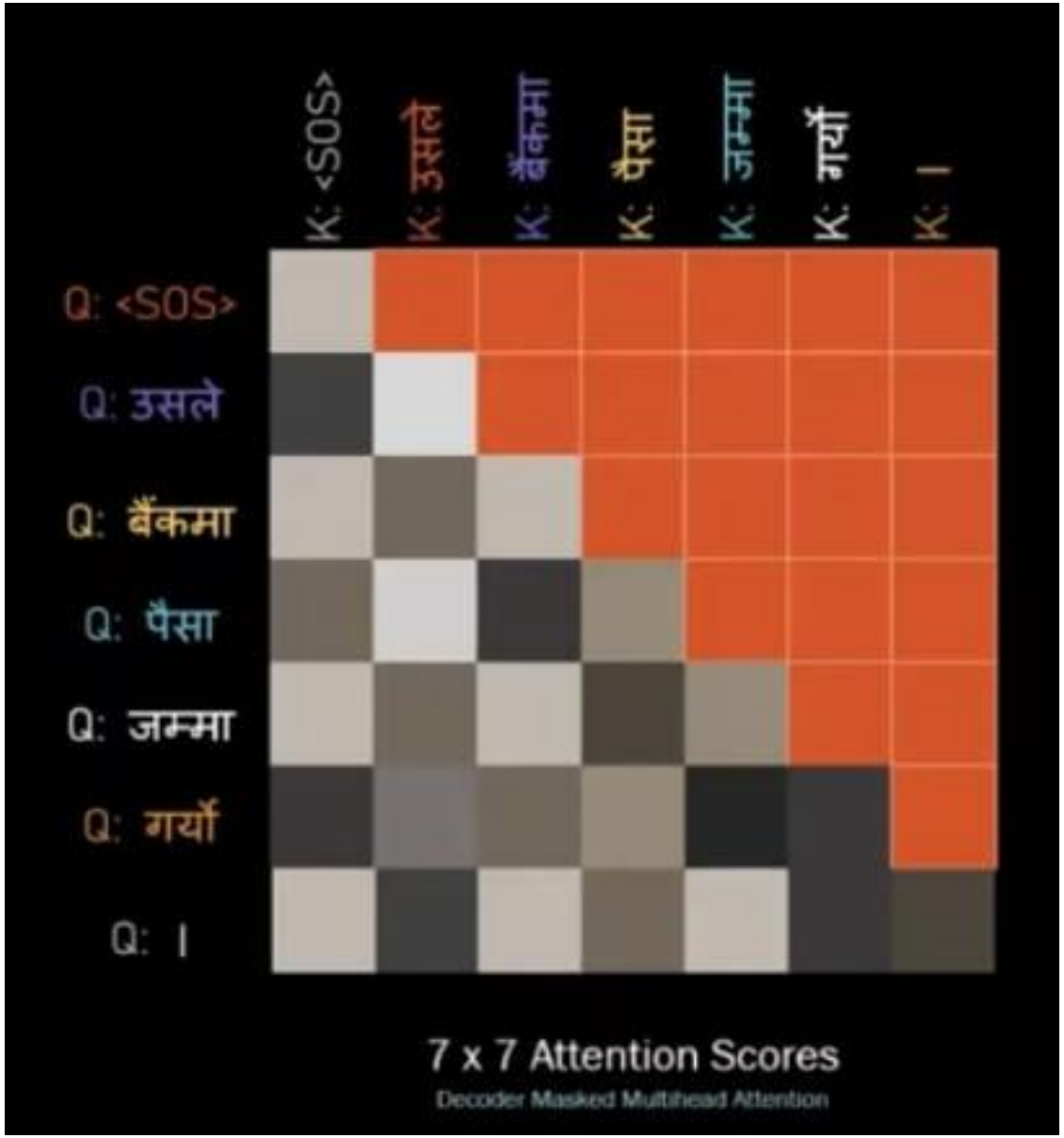
Credit: Grant Sanderson


[<SOS>] [He] [deposited] [money] [in] [the] [bank] [.] [<EOS>]
TOKENIZATION
<SOS> He deposited money in the bank. <EOS>

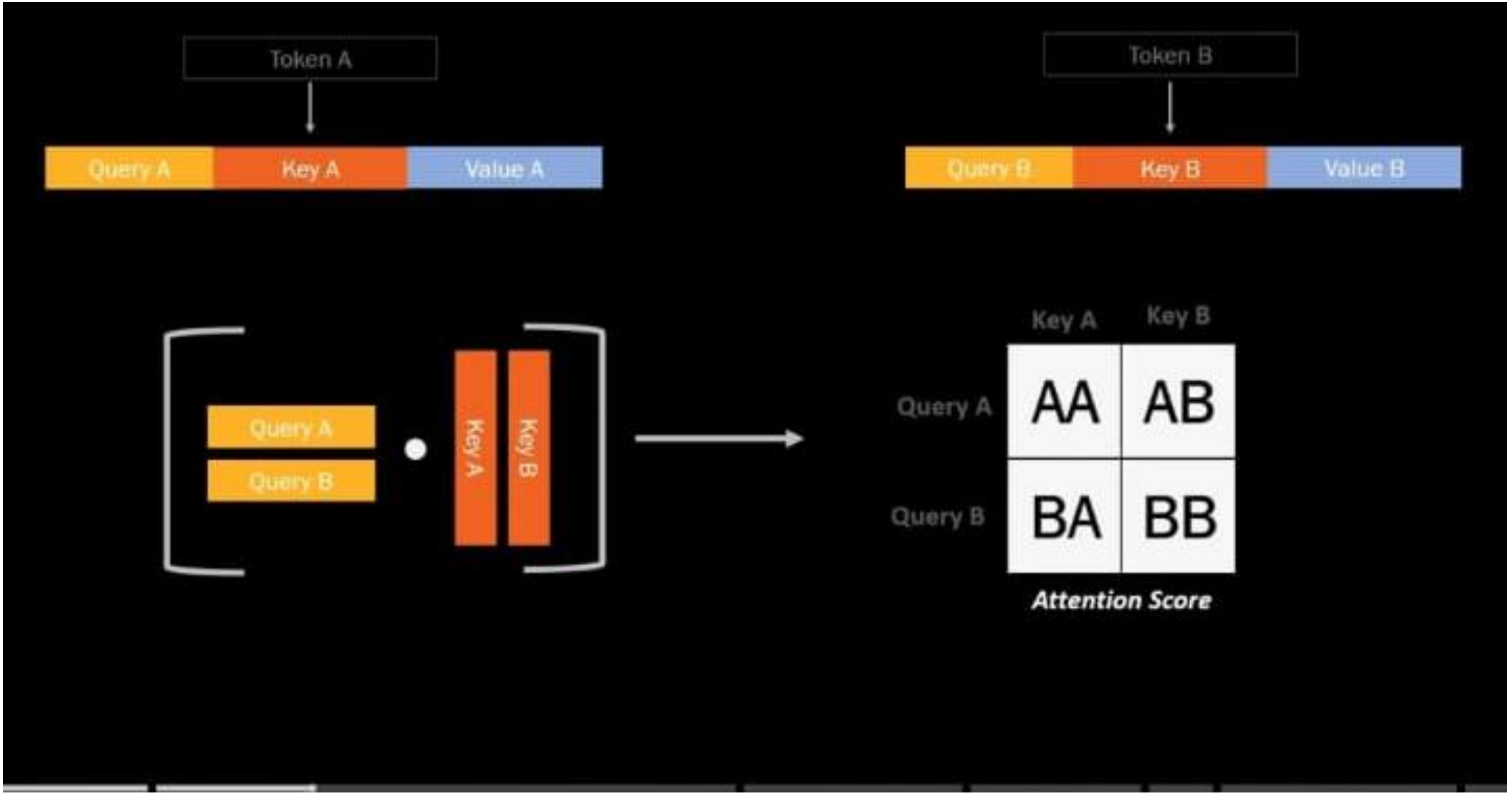




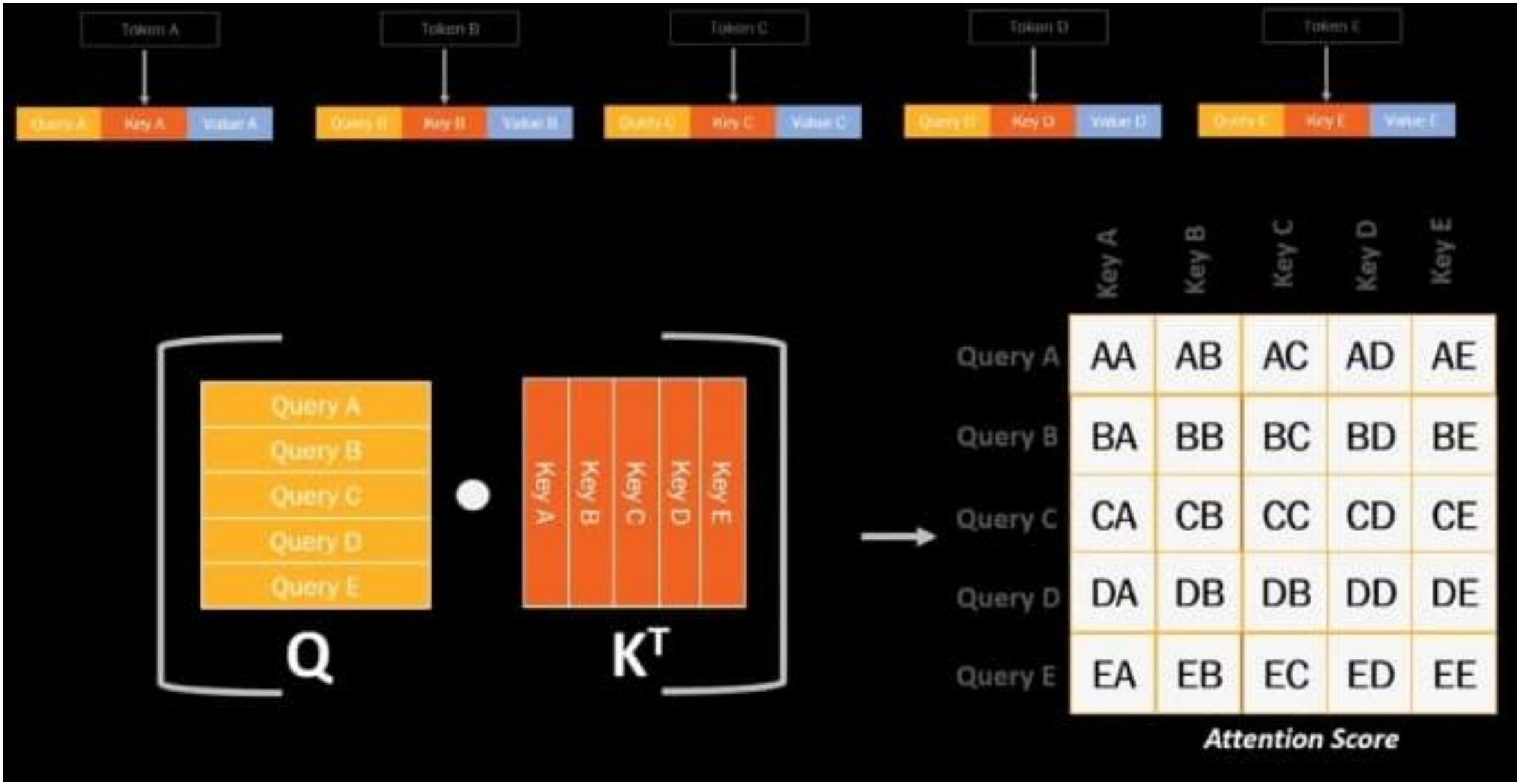
Credit: Grant Sanderson



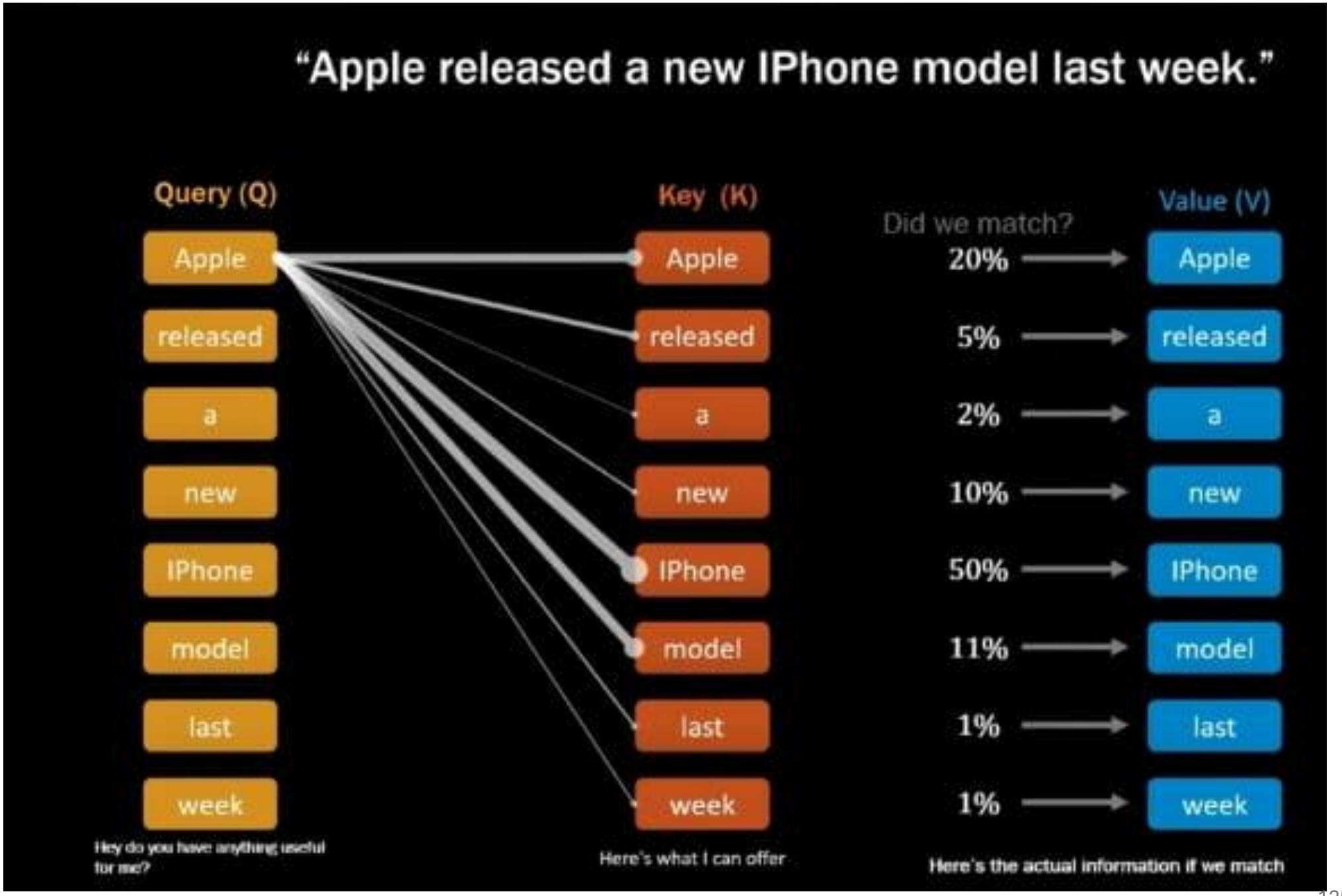
Credit: Grant Sanderson



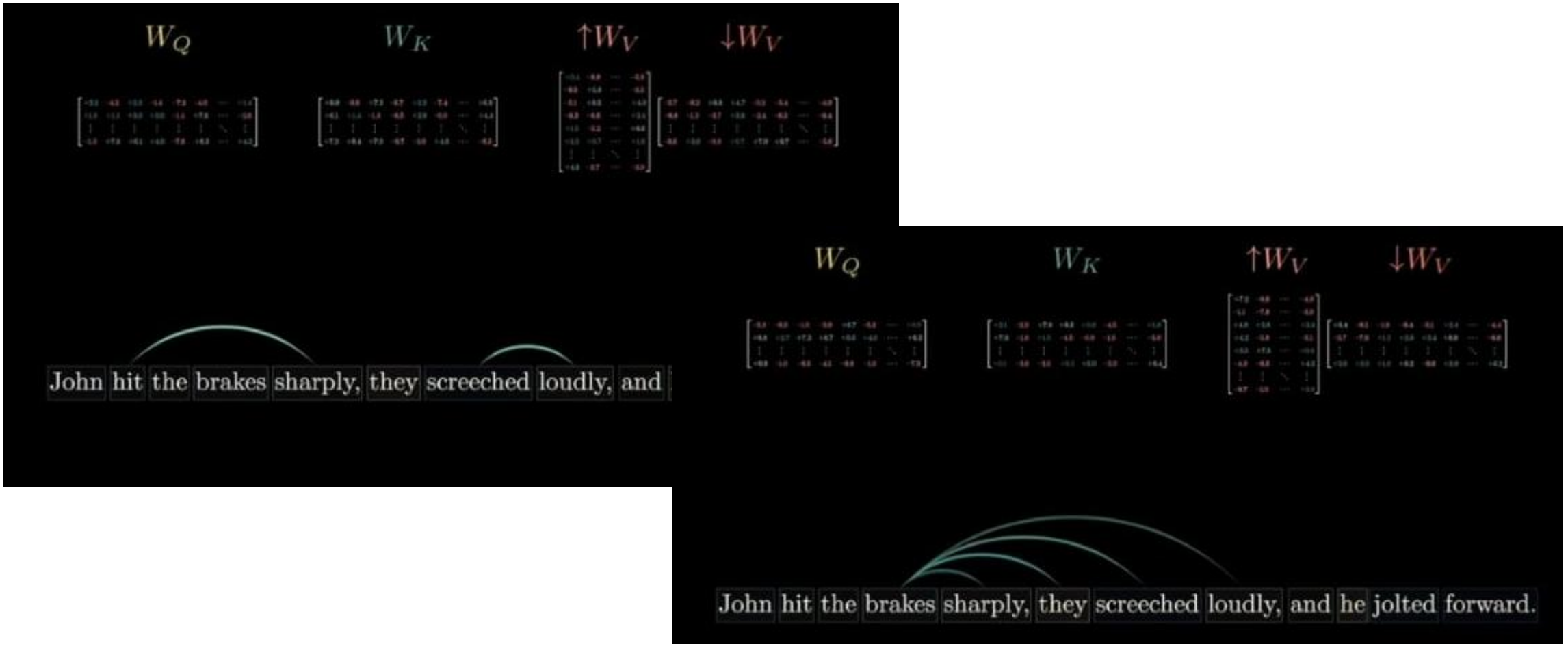
Credit: Grant Sanderson



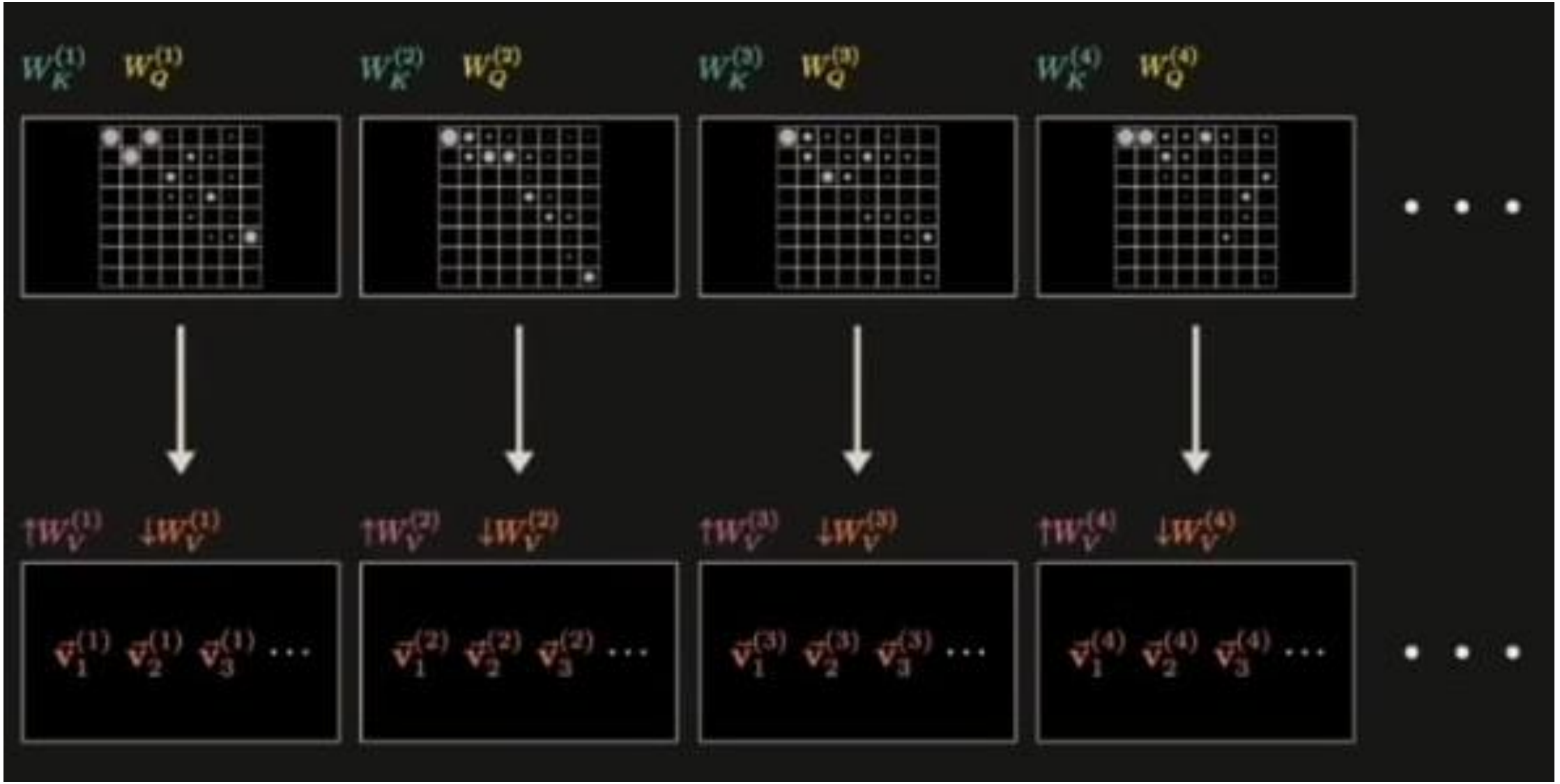
Credit: Grant Sanderson



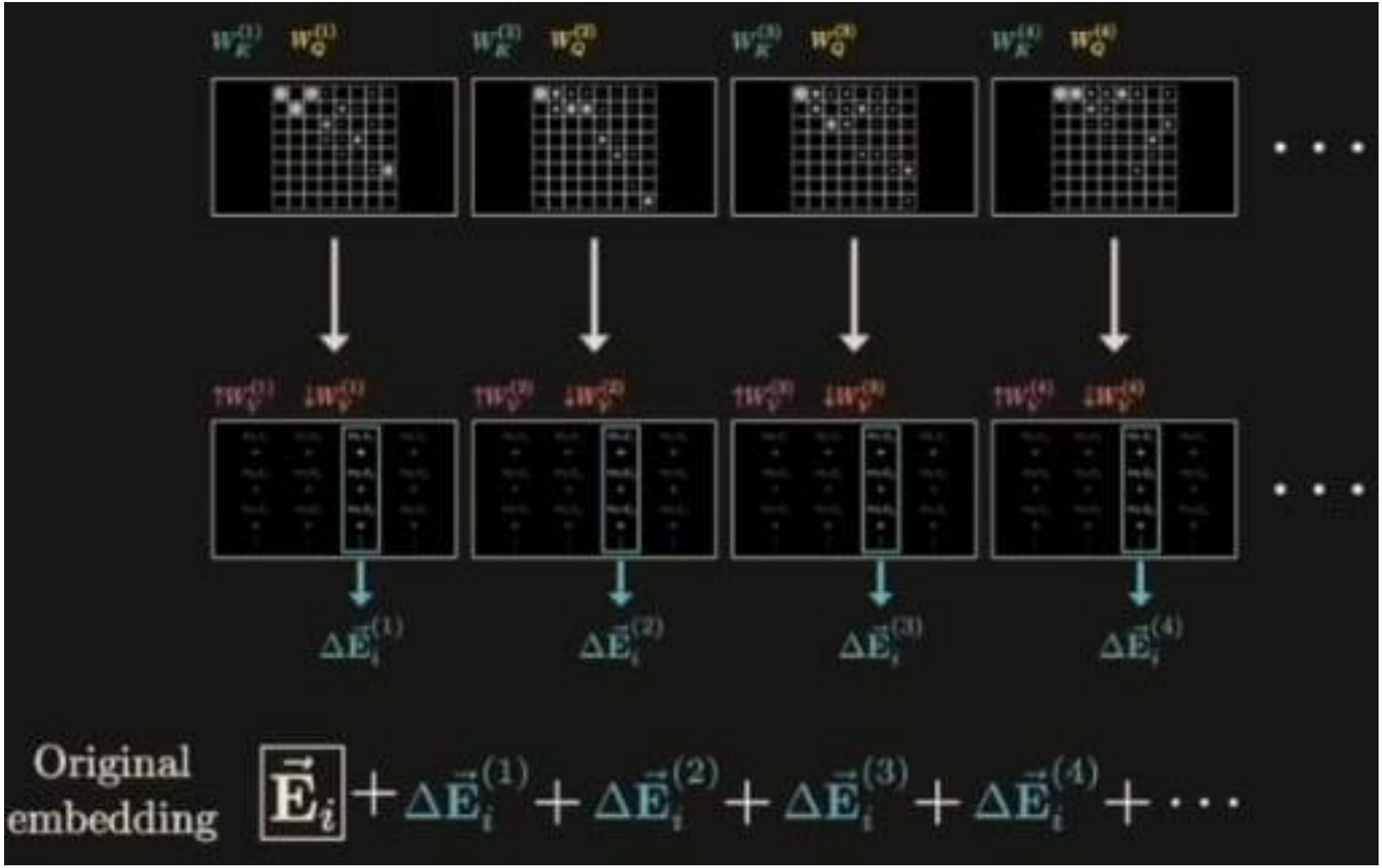
Credit: Grant Sanderson



Credit: Grant Sanderson



Credit: Grant Sanderson



Credit: Grant Sanderson

Transformer – Positional Encoding

- Transformers add sinusoid curves to the input, before the attention
  - Informs about relative position inside the sequence
  - Removes need for explicit recurrence patterns

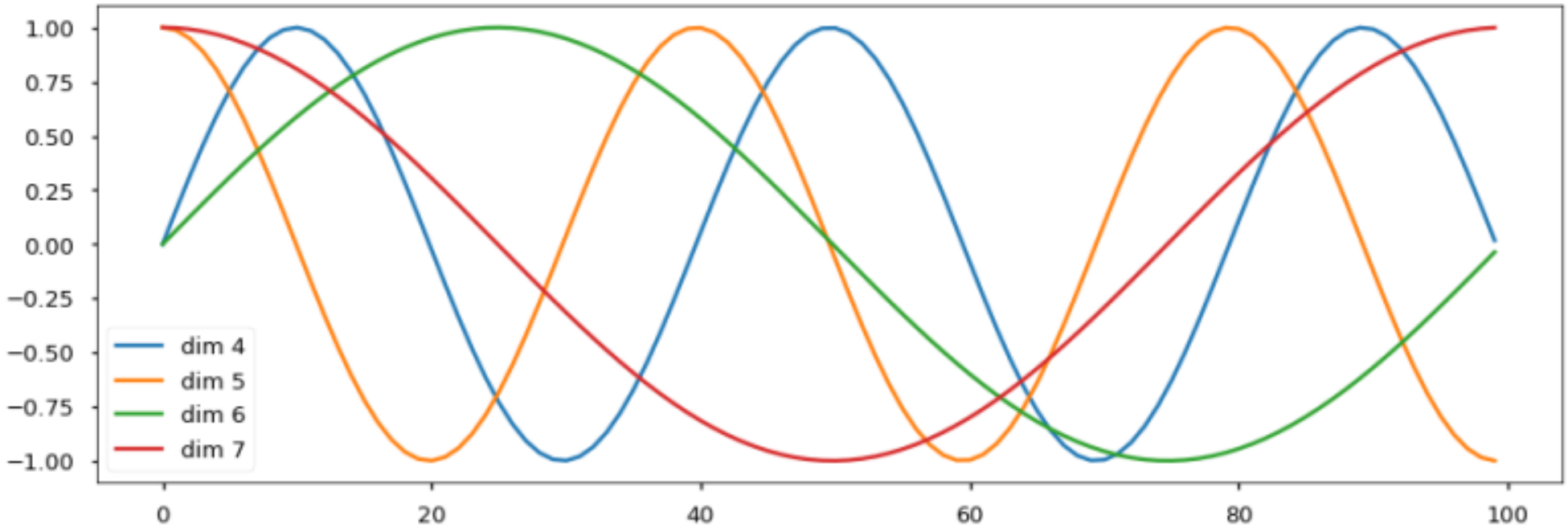
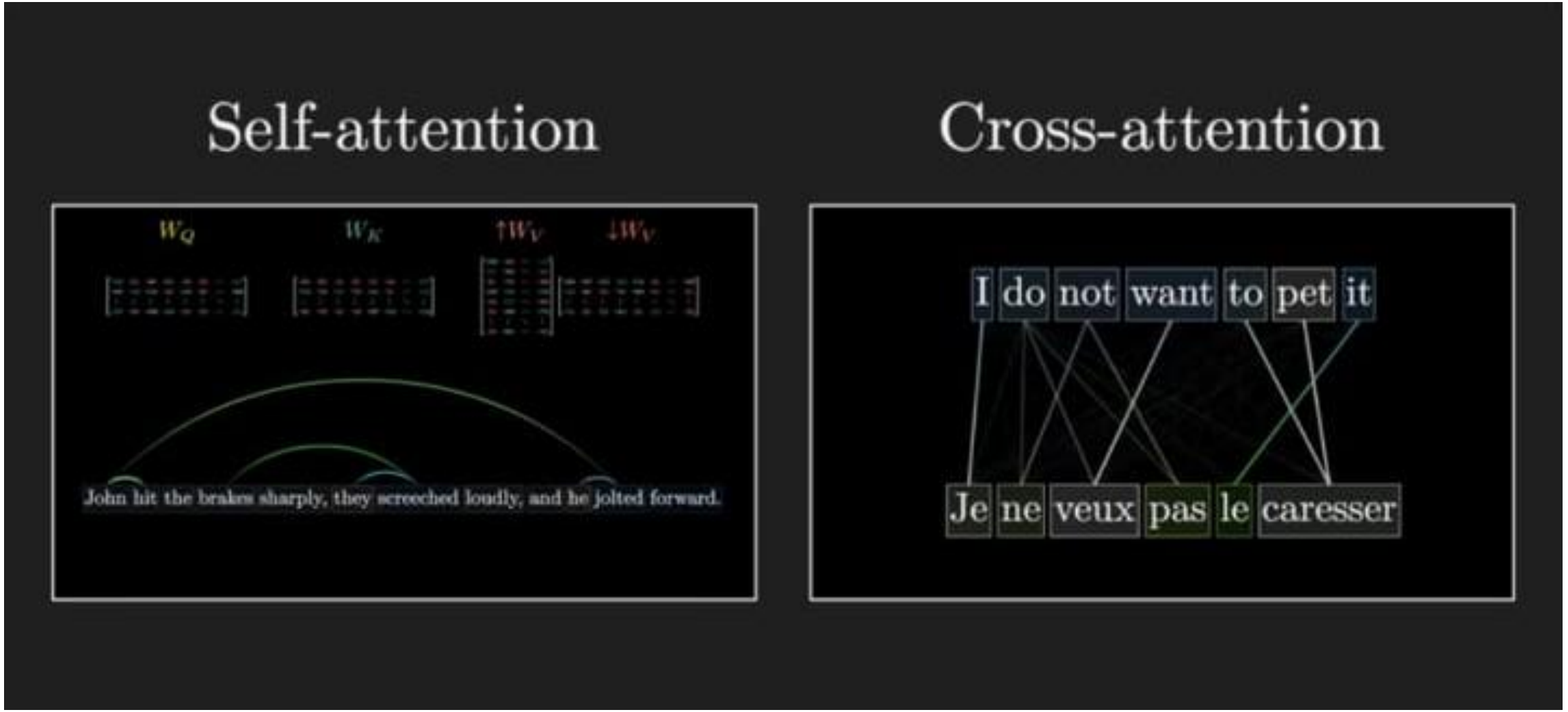


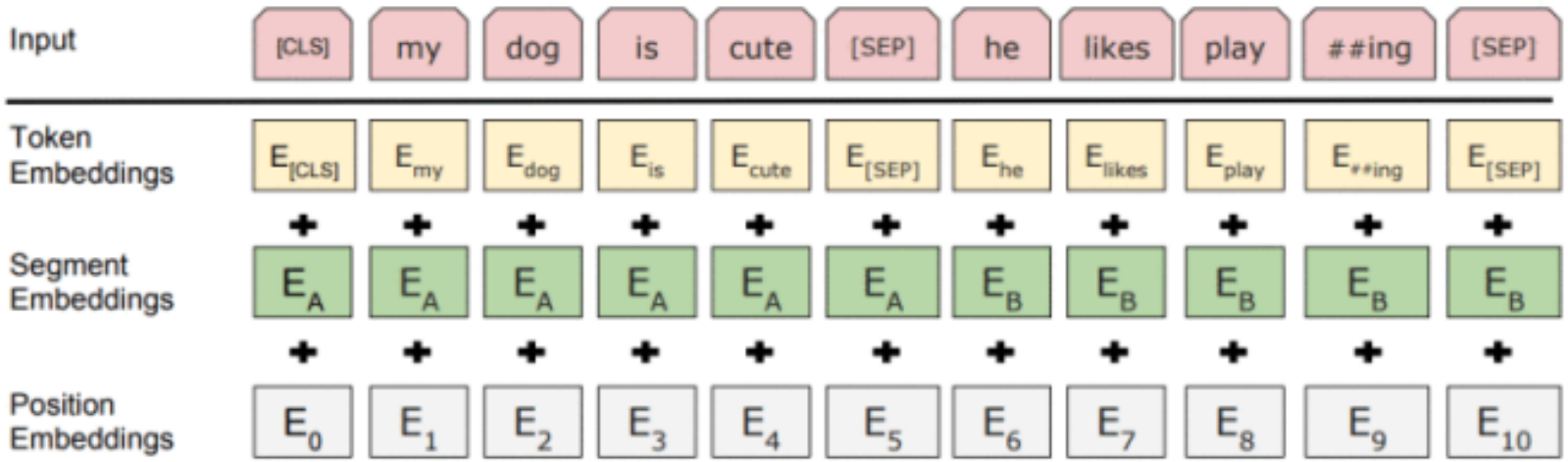
Figure from https://nlp.seas.harvard.edu/2018/04/03/attention.html



Credit: Grant Sanderson

BERT - Input

- Either one or two sentences, always prepended with [CLS]
  - BERT adds trained position embeddings & sequence embeddings



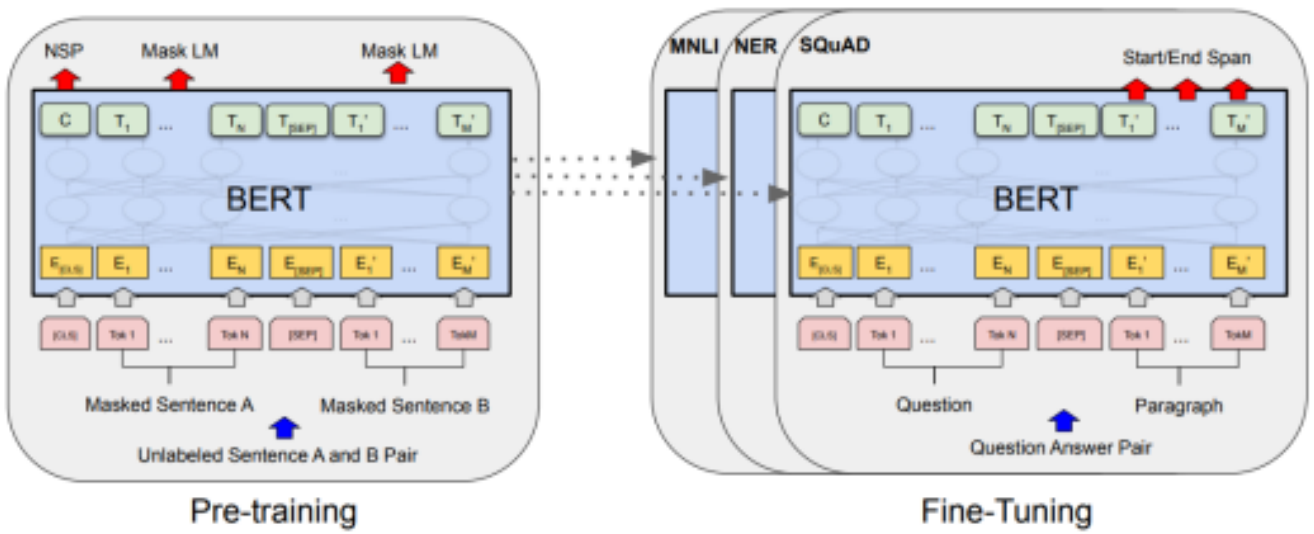
More info: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/

BERT - Model

- Model itself is quite simple: n Layers of stacked Transformers
  - Using LayerNorm, GeLU activations (like ReLU, but with a grace swing under 0)
  - Task specific heads on top to pool [CLS] or individual token representations
  - Every Transformer layer receives as input the output of the previous one
- The [CLS] token itself is only special because we train it to be
  - No mechanism inside the model that differentiates it from other tokens
- Novel contributions center around pre-training & workflow

BERT - Workflow

- Someone with lots of compute or time pre-trains a large model
  - BERT uses Masked Language Modelling [MASK] and Next Sentence Prediction [CLS]
- We download it and fine-tune on our task



BERT++

- Same as with Transformer variations, there are now many BERT variants
  - For many languages
  - Domains like biomedical publications
  - Different architectures, but similar workflow:
    Roberta, Transformer-XL, XLNet, Longformer …
- Main themes for adapted architectures:
  - Bigger
  - More efficient
  - Allowing for longer sequences (BERT is capped at 512 tokens in total)

Pre-Training Ecosystem

- With simple 1-word-1-vector embeddings (word2vec) sharing was as simples as a single text file containing both vocab + weights
  - We could simply load the weight matrix into bigger models
  - Mostly whitespace tokenization meant very little complexity

- BERT et al. re-use requires:
  - Exact model architecture (specific code and config) for hundreds of details
  - Weights for 100+ modules
  - Specific tokenizer rules for sub-word tokenization and special token handling
  - A single text file doesn't work here anymore …

Soooo many tasks are solvable with BERT

- Original BERT paper evaluates on:
  - GLUE, SQuAD, SWAG, CoNLL
- Now at ~18 thousand citations, we can assume some more are evaluated
  - As long as your text input is <512 tokens & you can pool the CLS token or learn per-term predictions you can use BERT
- HuggingFace model Hub alone provides out-of-the-box support for dozens of tasks & lists 200+ datasets used by the trained models.

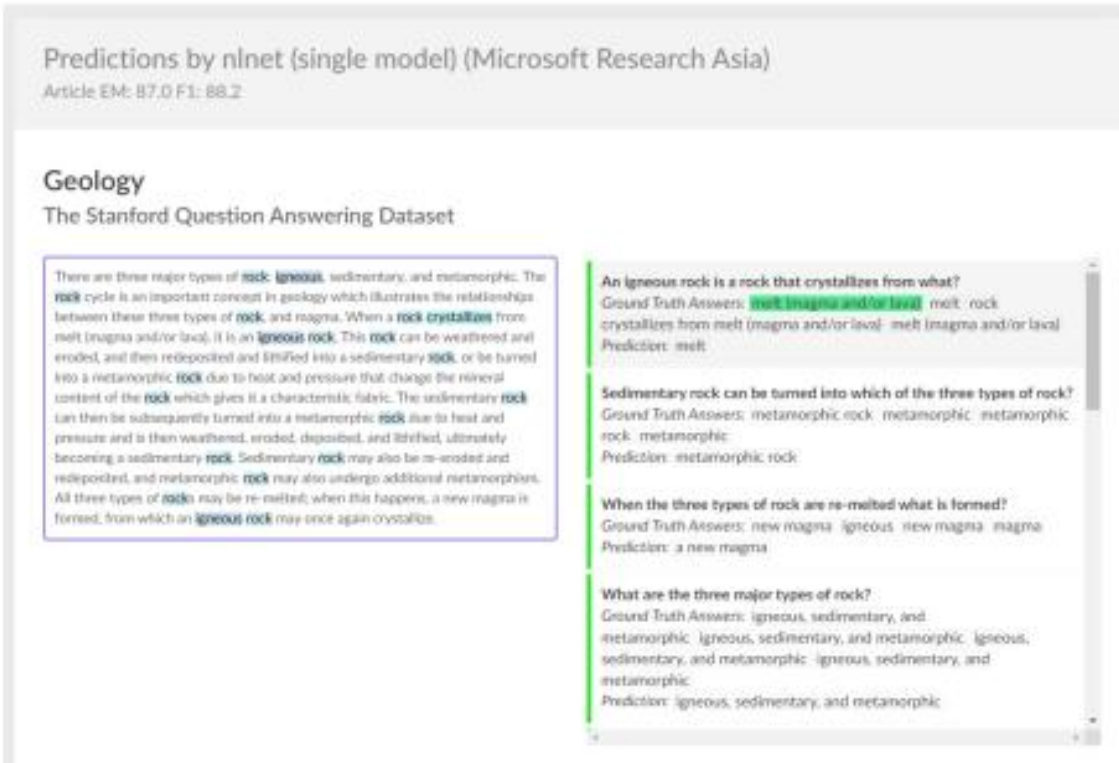
Extractive Question Answering

- Given a query and a passage/document:
  Select the words in the passage that answer the query
  - We want to select at least 1 span with a start and end position, that we then can extract
  - Use extracted text in highlighted UI (with surrounding text), chatbot, or audio-based assistant
  - Not perfect: query type must be specific to be answerable with fixed text

- Differs from *Generative* Question Answering
  - Models are tasked to create new text (with new words, more natural conversational style)
  - More complex, more potential for error and biases

Extractive QA: Datasets

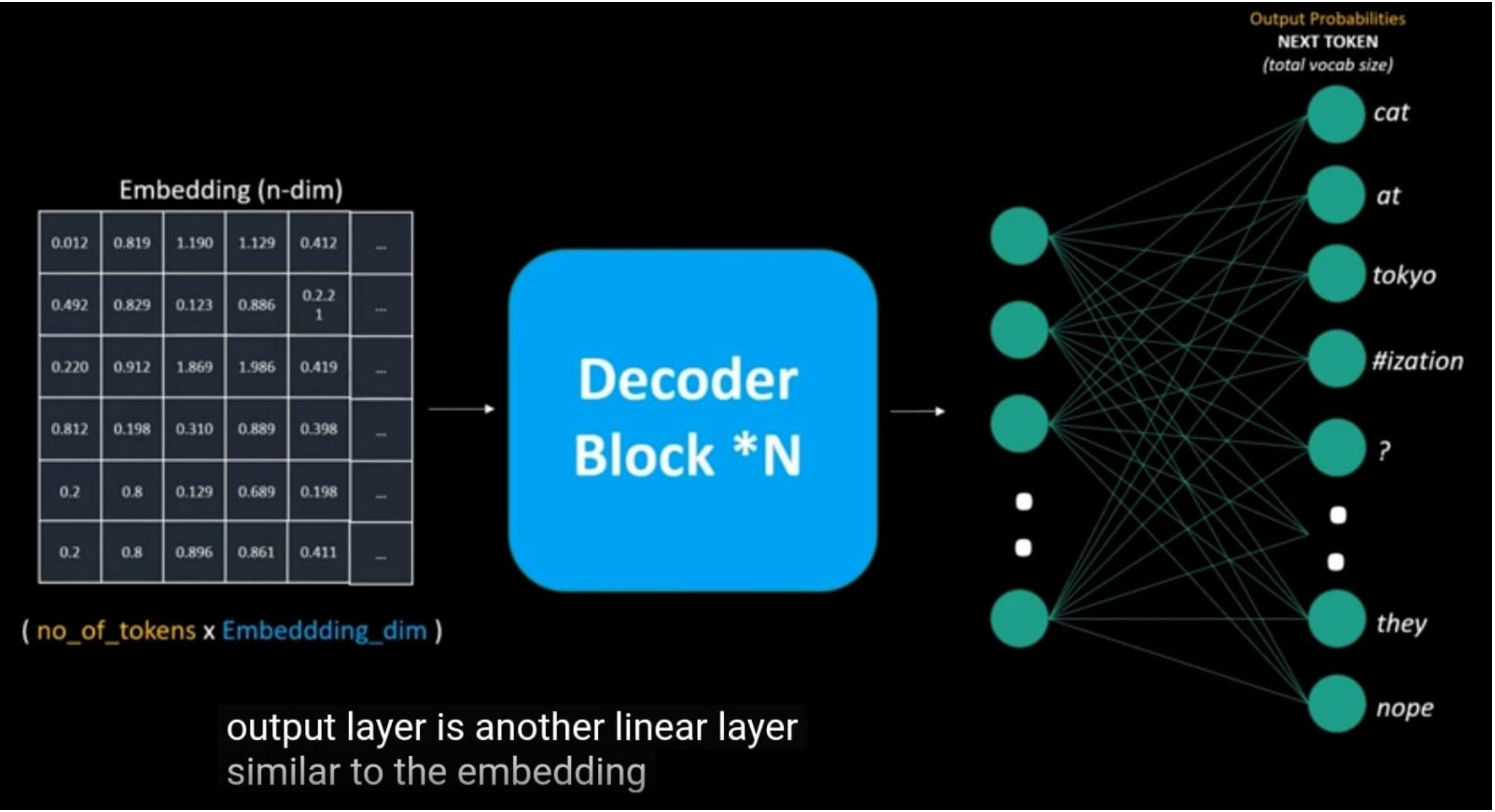
- Popular datasets include: SQuAD & NaturalQuestions
  - Both are based on Wikipedia text
  - SQuAD contains artificially created queries, NQ google search queries
  - Both come with fairly large training and evaluation sets
- Many pre-trained models are available for both



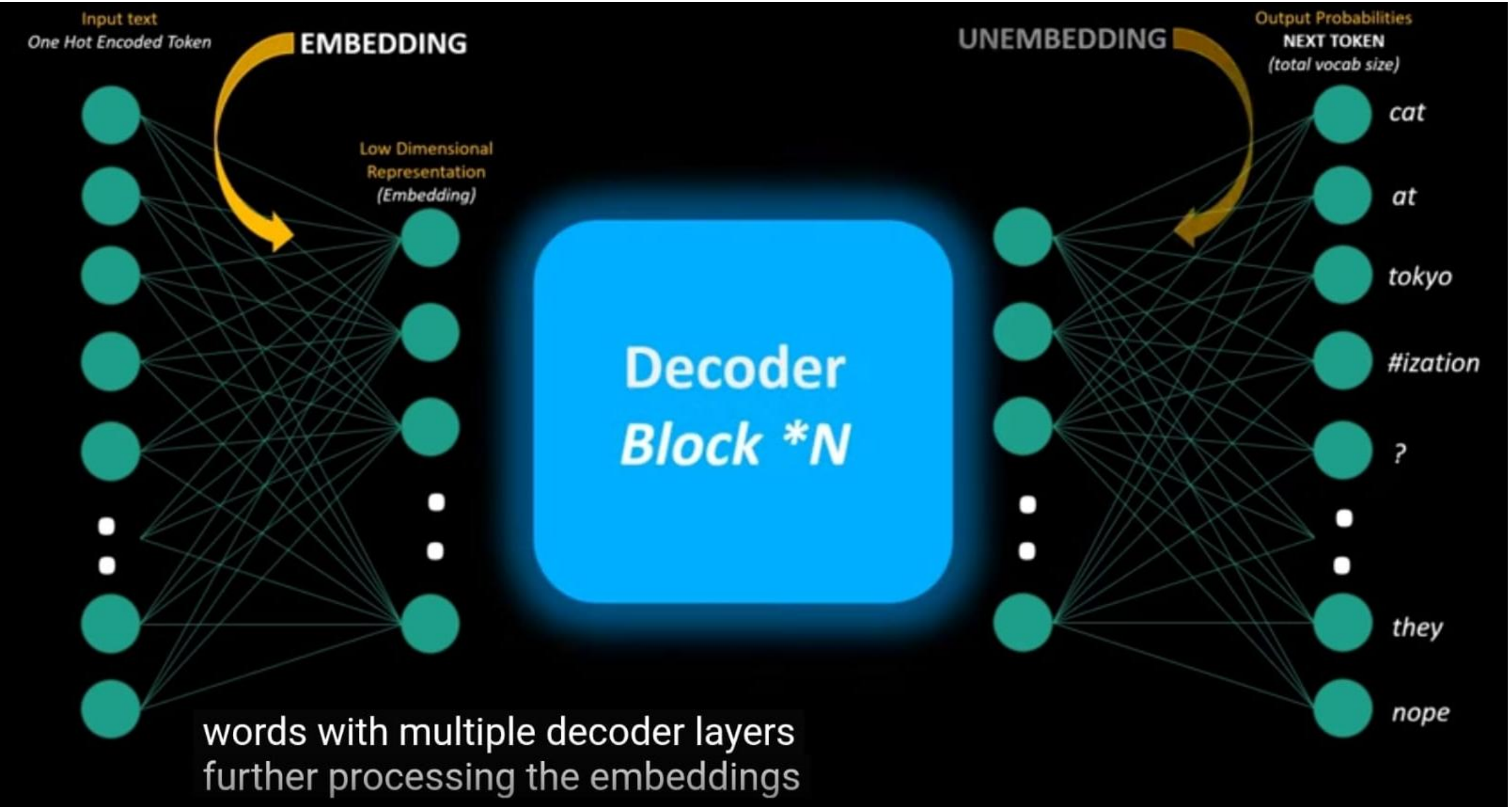
Example: https://rajpurkar.github.io/SQuAD-explorer/explore/v2.0/dev/

Extractive QA: Training

- For BERT we concatenate query and passage
  - With the pre-trained special tokens
- Per term output (of BERT) of the passage predicts if this token is a start or end token of the answer
  - End tokens are trained with gold-label start positions
  - Beam search can be used to find the best combination
- Loss is based on CrossEntropy of prediction vs ground-truth label
  - Potentially also includes a non-answerable prediction for the passage as a whole (SQuAD 2.0)



Credit: AbsolutelyAwesome (@Youtube)



Credit: AbsolutelyAwesome (@Youtube)

Encoder Input	Decoder Input
<SOS> Hello, how are you? <EOS>	<SOS> नमस्ते, तपाईंलाई कस्तो छ?
<SOS> What is your name? <EOS>	<SOS> तपाईंको नाम के हो?
<SOS> Where are you going? <EOS>	<SOS> तपाईं कहाँ जाँदै हुनुहुन्छ?
<SOS> She is going to the market to buy fruits. <EOS>	<SOS> उनी बजारमा फलफूल किन्न जाँदैछिन्।
<SOS> The children are playing football in the playground. <EOS>	<SOS> बच्चाहरू खेलमैदानमा फुटबल खेलिरहेका छन्।
<SOS> The teacher is explaining the lesson very clearly <EOS>	<SOS> शिक्षकले पाठलाई धेरै स्पष्ट रूपमा सम्झाइरहेका छन्।

Credit: AbsolutelyAwesome (@Youtube)

d_{model} = Model Embedding = 512

h = number of Attention Heads = 4

$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$

$d_k = dv = (d_{model} / h) = (512 / 4) = 128$

$W_i^K \in \mathbb{R}^{d_{model} \times d_k}$

d_k = Projection Dimension of Key and Query = 128

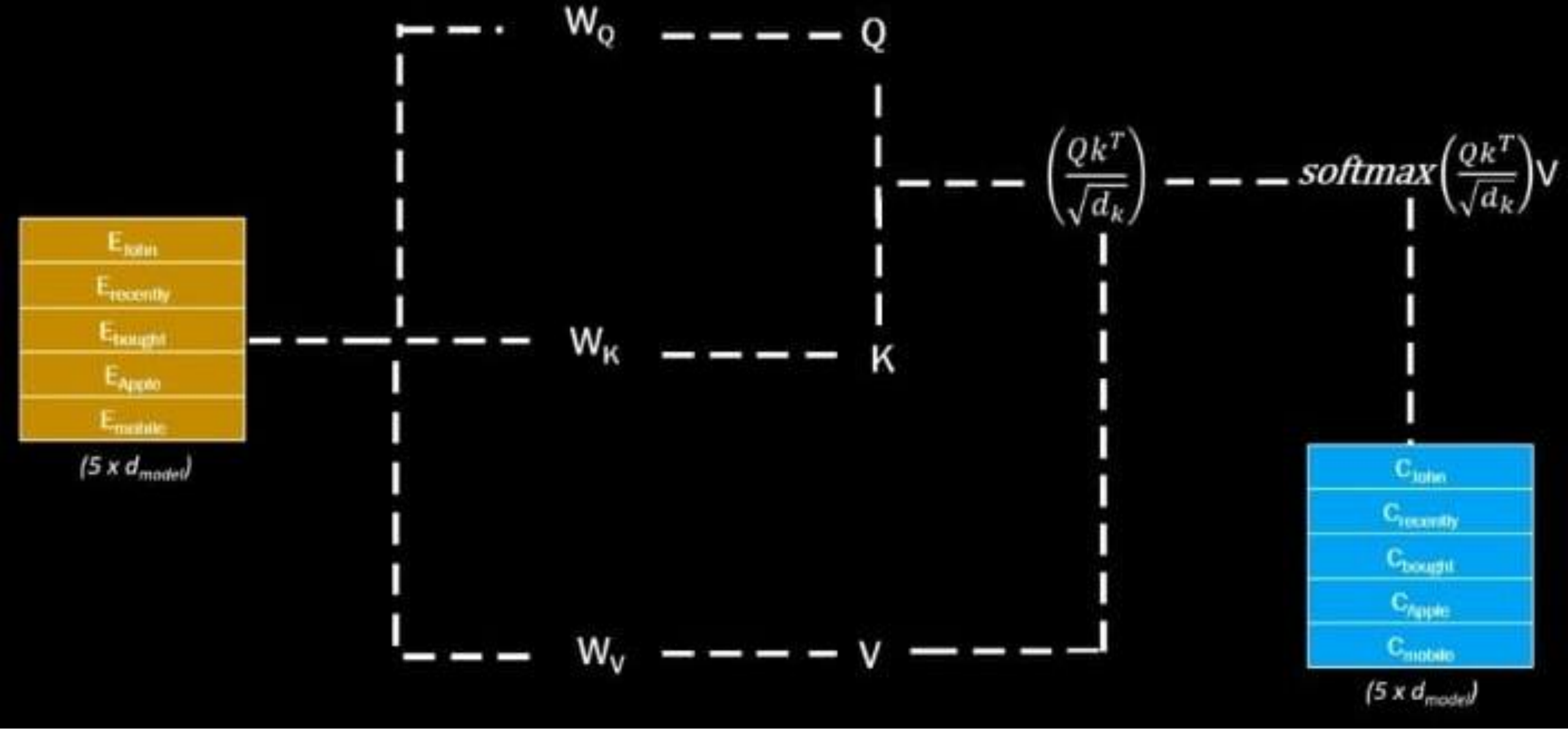
$W_i^V \in \mathbb{R}^{d_{model} \times d_v}$

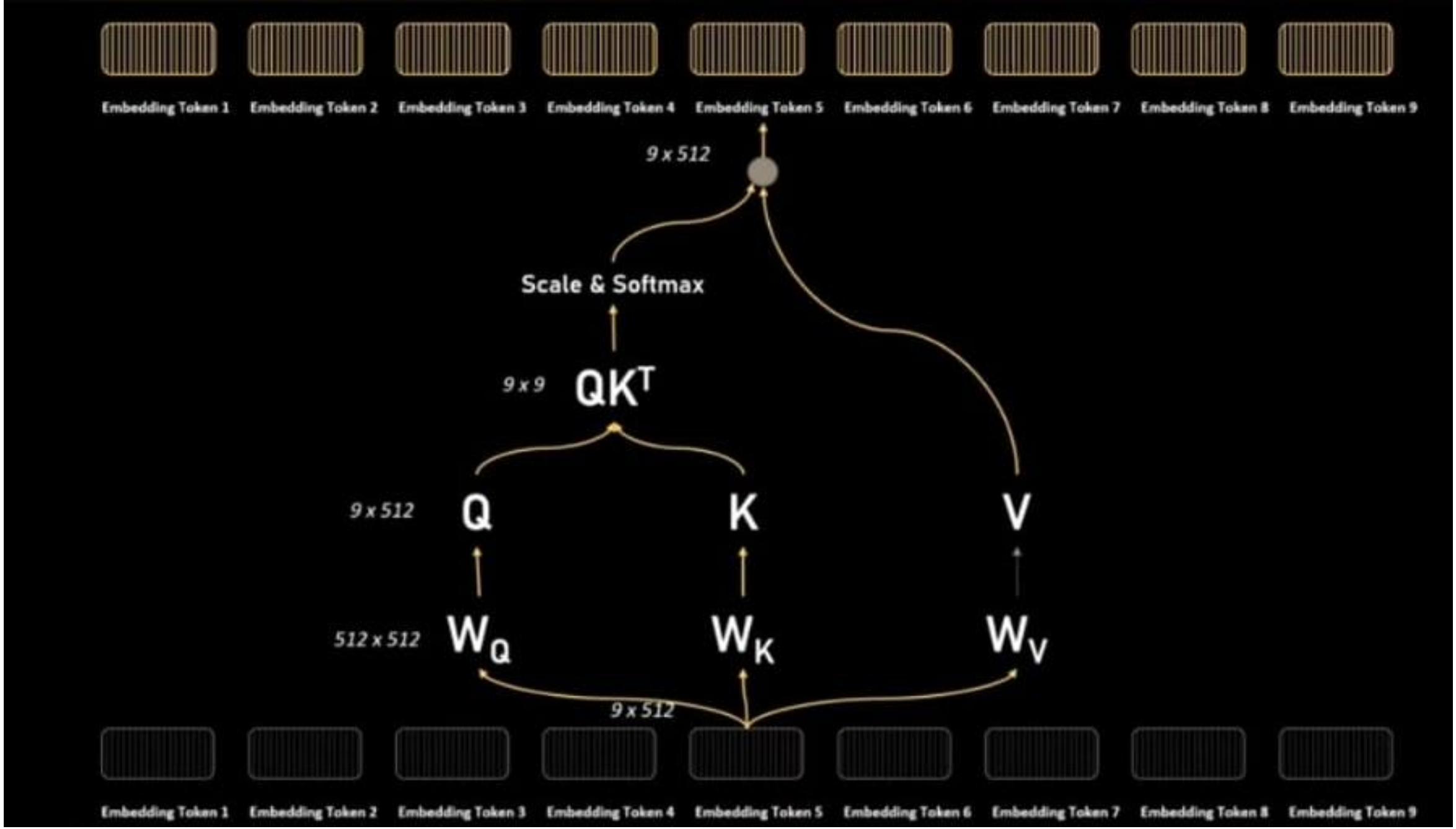
d_v = Projection Dimension of Value = 128

$W^O \in \mathbb{R}^{hd_v \times d_{model}}$

W^O = Output Projection (4x128 * 512)

512 additionally we have another new weight Matrix W


W_Q
Q
$\left(\frac{Qk^T}{\sqrt{d_k}}\right)$
$softmax\left(\frac{Qk^T}{\sqrt{d_k}}\right)V$
E_{John}
$E_{recently}$
E_{bought}
E_{Apple}
E_{mobile}
W_K
K
$(5 \times d_{model})$
C_{John}
$C_{recently}$
C_{bought}
C_{Apple}
C_{mobile}
W_V
V
$(5 \times d_{model})$


Embedding Token 1
Embedding Token 2
Embedding Token 3
Embedding Token 4
Embedding Token 5
Embedding Token 6
Embedding Token 7
Embedding Token 8
Embedding Token 9
9 x 512
Scale & Softmax
9 x 9
QK^T
9 x 512
Q
K
V
512 x 512
W_Q
W_K
W_V
9 x 512
Embedding Token 1
Embedding Token 2
Embedding Token 3
Embedding Token 4
Embedding Token 5
Embedding Token 6
Embedding Token 7
Embedding Token 8
Embedding Token 9

Until the next time ☺