الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

المحاضرة الأولى	كلية الهندسة	الذكاء الصنعي العملي

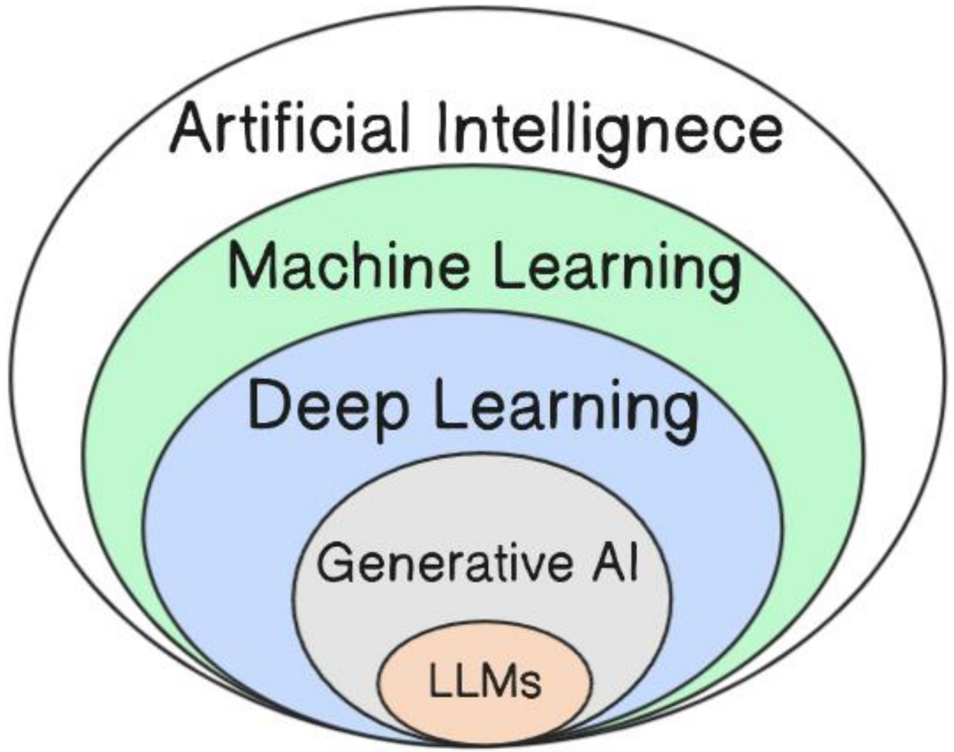
Introduction to Generative AI & LLMs
Word Embeddings 1

د. رياض سنبل

What is Generative AI?

- Generative AI is a branch of artificial intelligence focused on **creating new content** from learned patterns in data. Generative AI describes a category of capabilities within AI that create **original content**.

- It includes models that generate:
  - **Text** (ChatGPT, Claude, Bard)
  - **Images** (DALL-E, Stable Diffusion)
  - **Music & Audio** (Jukebox, VALL-E)
  - **Code** (GitHub Copilot, Code Llama)



Evolution of AI & NLP

- Early AI models (rule-based systems) → Statistical NLP → Deep Learning-based NLP
- **Major breakthroughs:**
  - 2013: Word Embeddings (Word2Vec, GloVe, FastText) <= Today
  - 2017: Transformers (Attention Is All You Need)
  - 2018: BERT (Bidirectional Transformers)
  - 2020–Present: GPT-3, GPT-4, LLaMA, Claude, Mistral

Language Models

- LMs model the probability distribution of token sequences in the language.
  - Word sequences, if words are the tokens
- Can be used to:
  - Compute the probability of a given token sequence
  - Generate sequences from the distribution of the language

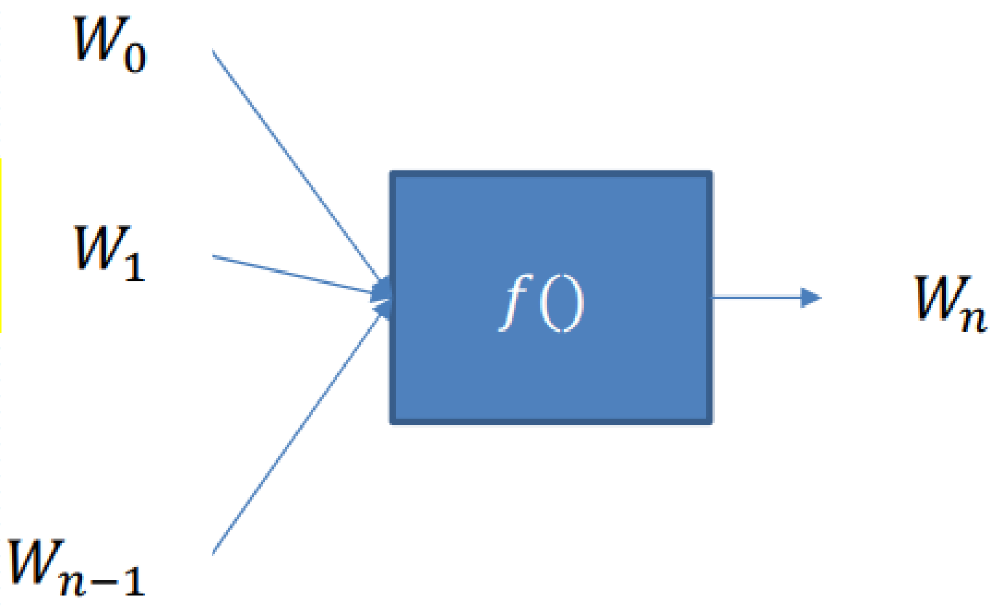
$$\boxed{\begin{aligned} P(w_1\, w_2\, w_3\, w_4\, ...) = P(w_1)\, P(w_2|w_1) \\ P(w_3|w_1\, w_2)\, P(w_4|w_1\, w_2\, w_3)\ldots \end{aligned}}$$

- The actual target is to model the probabilities of entire word sequences
- However, we typically use Bayes rule to compute this incrementally
  - Language models generally perform next symbol prediction

Language modelling - Next Symbol Prediction

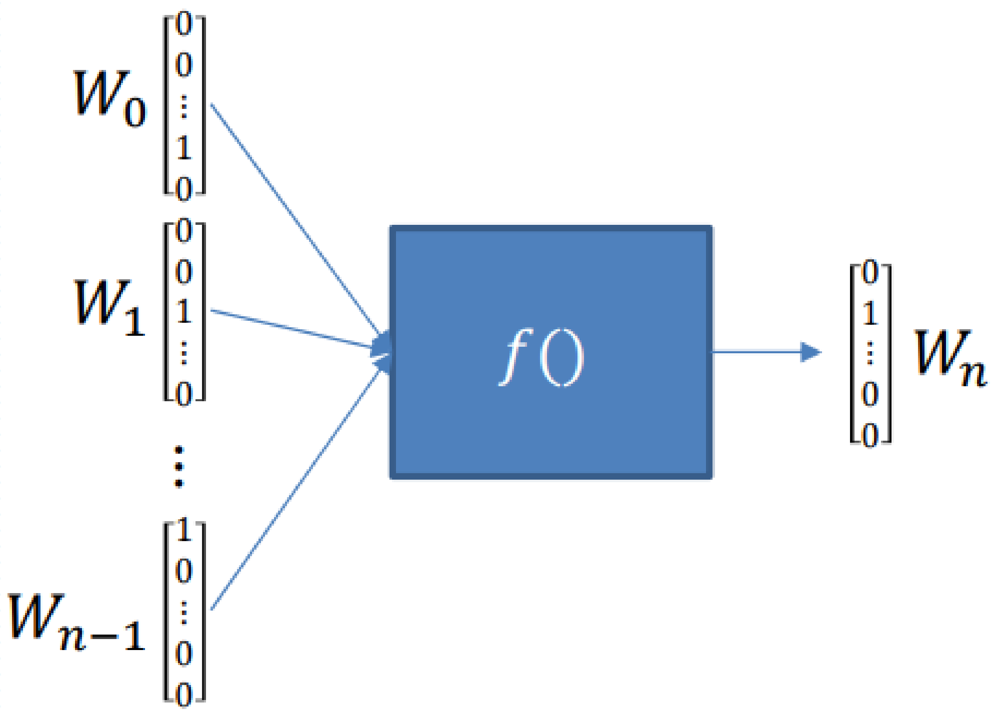
Problem: Given a sequence of words (or characters) predict the next one

But How to Represent words?



Language modelling - Next Symbol Prediction

Problem: Given a sequence of words (or characters) predict the next one



Represent words as one-hot vectors

- One challenge is the **size of the representation**.
- Possible Solutions:
  - Its common to **discard infrequent words** occurring less than e.g. 5 times.
  - one can also use the **feature selection methods**: mutual information and chi-squared tests.
  - **Preprocessing** steps: stemming.
- What should we **include** in this dictionary:
  - Bag-of-words
  - OR: Bag-of-sences, or bag-of-characters, etc.

Represent words as one-hot vectors

- **<u>Word order is lost</u>**, the sentence meaning is weakened:
  - The mat sat on the cat vs The cat sat on the mat.
  - word order is important, especially the order of nearby words.
- Possible Solutions:
  - **N-grams capture this**, by modeling tuples of consecutive words.
  - Typically, its advantages to use multiple n-gram features in machine learning models with text, e.g. unigrams + bigrams (2-grams) + trigrams (3-grams).

Notice how even these short n-grams "make sense" as linguistic units. For the other sentence we would have different features:

Sentence: *The mat sat on the cat*

2-grams: *the-mat, mat-sat, sat-on, on-the, the-cat*

We can go still further and construct 3-grams, Which capture still more of the meaning:

Sentence: *The mat sat on the cat*

3-grams: *the-mat-sat, mat-sat-on, sat-on-the, on-the-cat*

Represent words as one-hot vectors

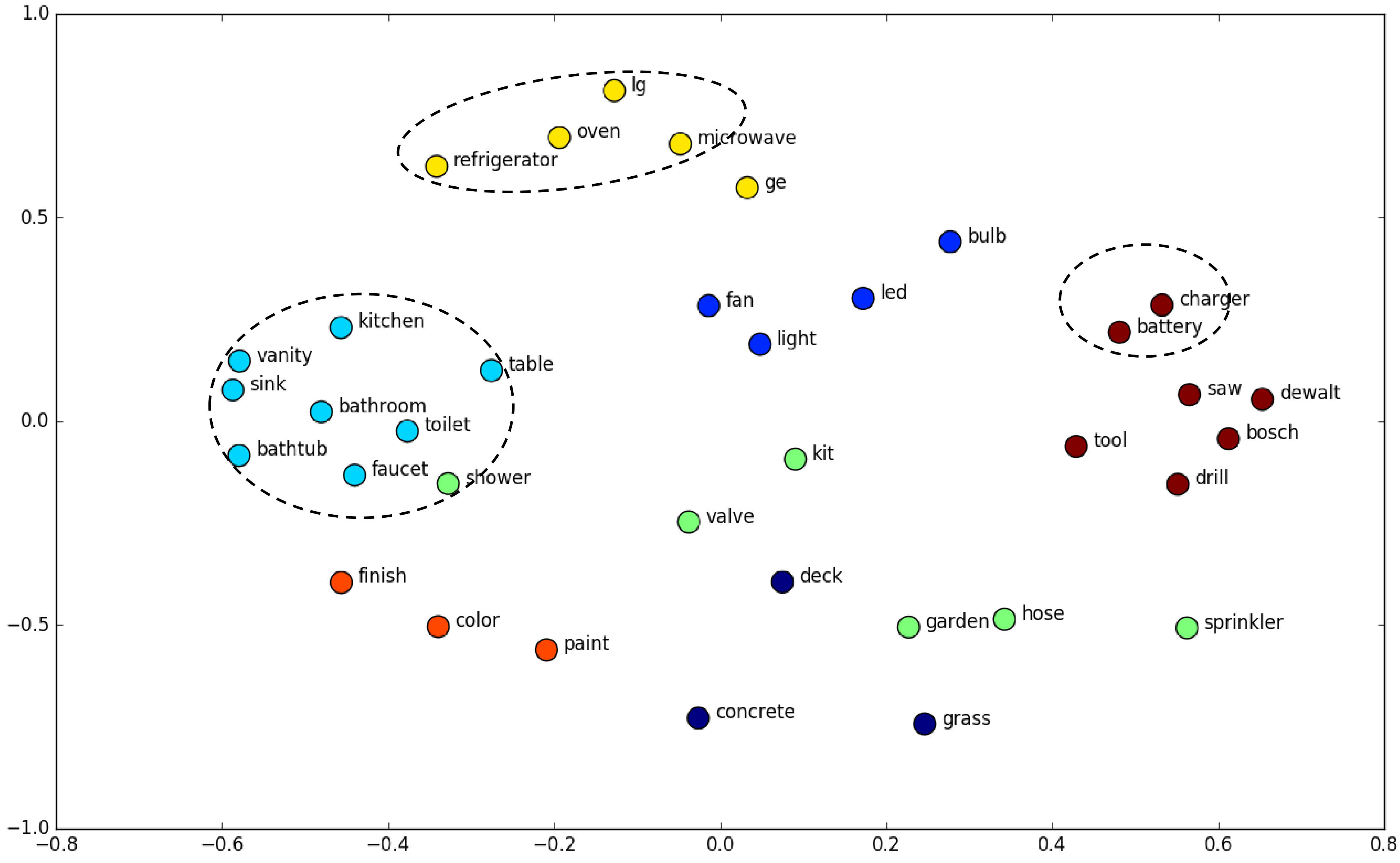
OR... Representing words as **low-dimensional dense vectors**

Word Embedding

Vector Embedding of Words

- A word is represented as a vector.
- Word embeddings depend on a notion of ***word similarity***.
- A very useful definition is paradigmatic similarity:
  - ***Similar words*** occur in ***similar contexts.*** They are ***exchangeable.***
  - Yesterday { POTUS / The President / Trump } called a press conference.
  - "POTUS: President of the United States."

The “Good” Embedding?



Good Embedding

≈

Word similarity

$$\cos(w_x, w_y) = \frac{w_x \cdot w_y}{||w_x||\ ||w_y||}$$

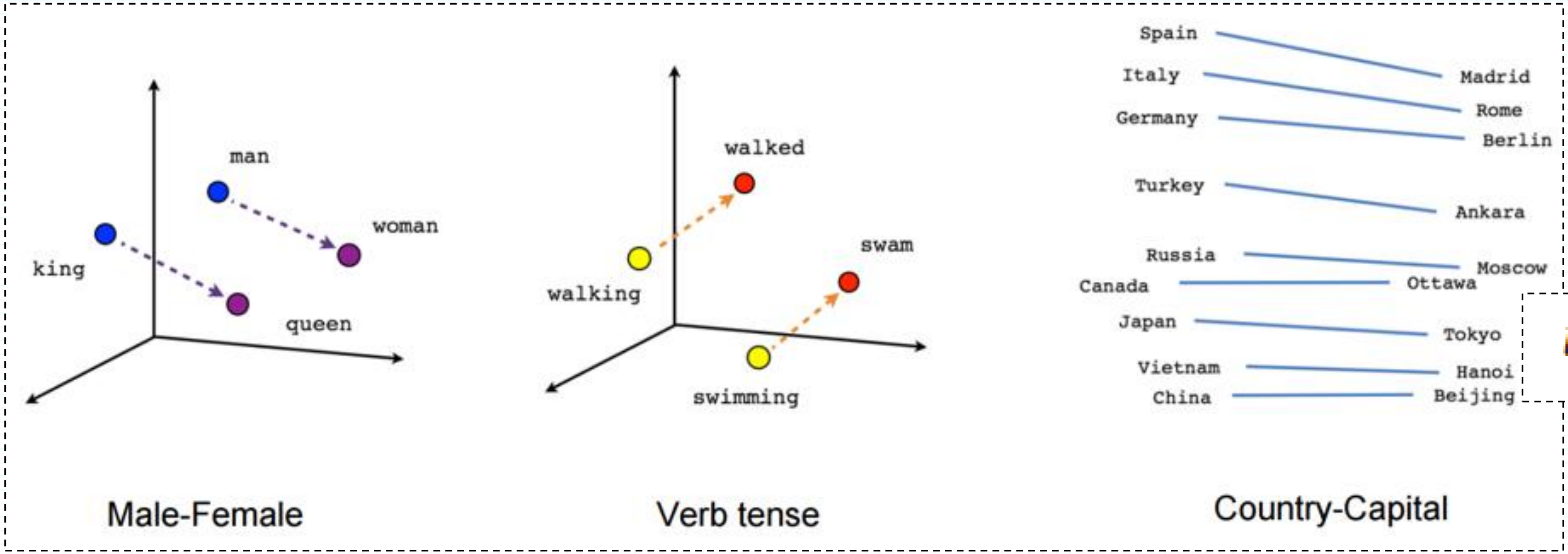
The “Good” Embedding?

Good Embedding

≈

Word analogy

$$b^* = \underset{b'}{\mathrm{argmax}}(\cos(b', a^* - a + b)),$$



- vector[Queen] ≈ vector[King] - vector[Man] + vector[Woman]
- vector[Paris] ≈ vector[France] - vector[Italy] + vector[Rome]
  - This can be interpreted as “France is to Paris as Italy is to Rome”.

Word Embedding

- An embedding maps each word to a point in a much smaller m-dim space (e.g. 300 values)
- Two approaches:
  - **Learn the embedding jointly with your main task:**
    - An embedding layer with m hidden nodes to map word IDs to an m-dim vector.
    - Add your hidden layer and output layer, learn weights end-to-end with SGD.
  - **Use pre-trained embedding:**
    - Usually trained on another, much bigger dataset.
    - Freeze embedding weights to produce simple word embedding.

Training Embedding Layer

- Input layer use fixed length document (e.g. 100 nodes for 100 word IDs).
  - Pad with 0's is doc is shorter.
- Add an embedding layer to learn embeddings:
  - (1) Represent every word as an n-dim one hot encoding BOW.
  - (2) Learn an m-dim embedding using m-hidden nodes.
    - Learn weight matrix W(n*m) to map one hot encoded word to embedding.
  - (3) Use Linear activation function => $X_{embed} = W. X_{orig}$
- Add a layer to map word embedding to desired output.
- Learn all weights from labeled data.

Pre-trained embeddings

- More data => better embeddings BUT also **more labels**!
- **Solution**: self-supervised learning
  - Given a word, predict the surrounding words.
- Most common approaches:
  - **Word2vec:** learn neural embedding for word based on surrounding words.
  - **GloVe (Global Vector):** Count co-occurences of words in matrix.
  - **FastText:** learns embedding for character n-gram
  - Language models: learn context-dependenct embedding.
    - **BERT**, **ELMO**, **GPT3**