95-891:
Introduction to Artificial Intelligence

Session 12: Deep Learning for NLP

Yubo Li
yubol@andrew.cmu.edu

Feb 20, 2025

Agenda

- A Framework for NLP Systems
- Word Representation
- Language Modeling
- Recurrent Neural Network (RNN)
- Transformer
  - Word Embedding
  - Positional Encoding
  - Attention

A Framework for NLP Systems

A Framework for NLP Systems

- Create a function to map an input *X* into an output *Y*, where *X* and/or *Y* involve language.

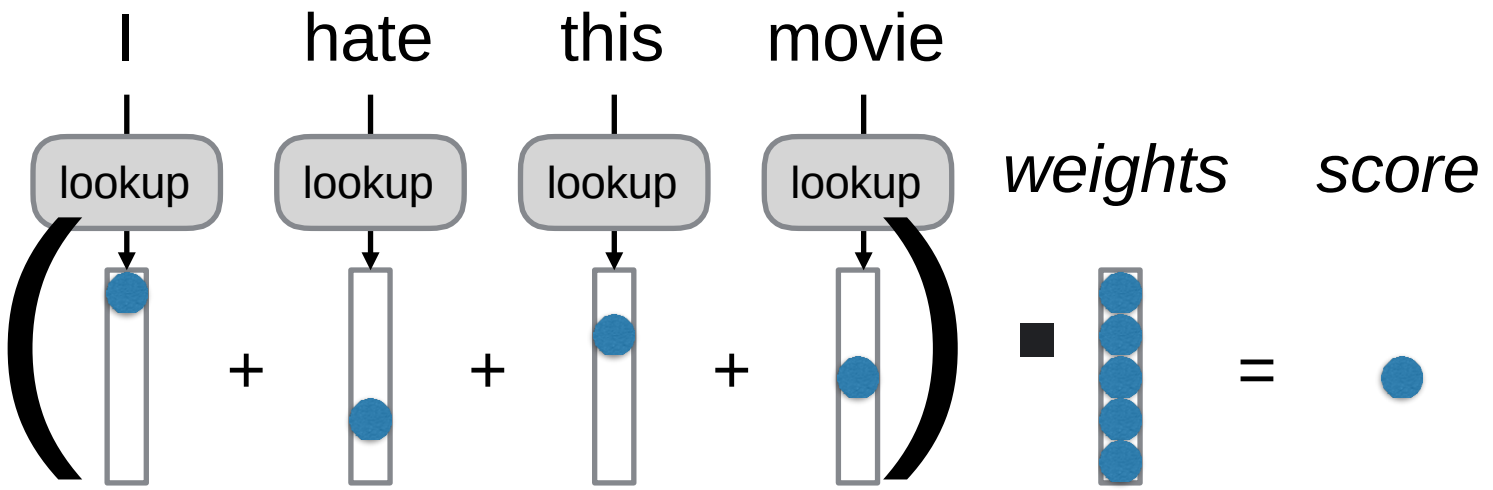
A Framework for NLP Systems

- Create a function to map an input *X* into an output *Y*, where *X* and/or *Y* involve language.

Input *X*	Output *Y*	Task
Text	Continuing Text	Language Modeling
Text	Text in Other Language	Translation
Text	Label	Text Classification
Text	Linguistic Structure	Language Analysis
Image	Text	Image Captioning

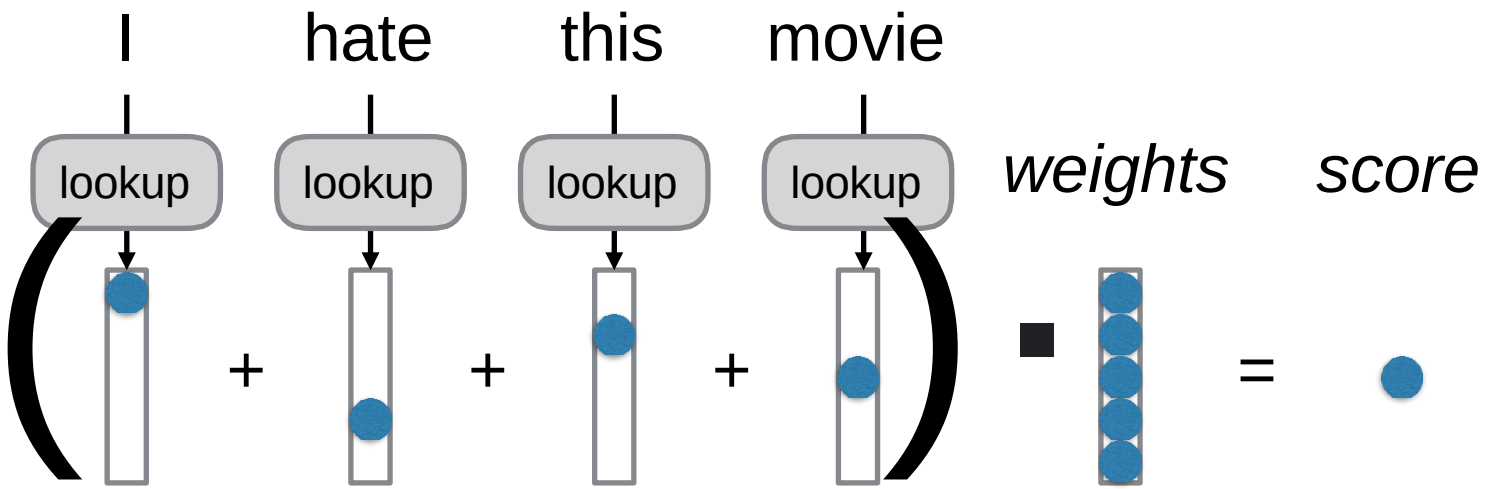
Word Representation

Word representation – Bag of Words



Features *f* are based on word identity, weights *w* learned

Word representation – Bag of Words

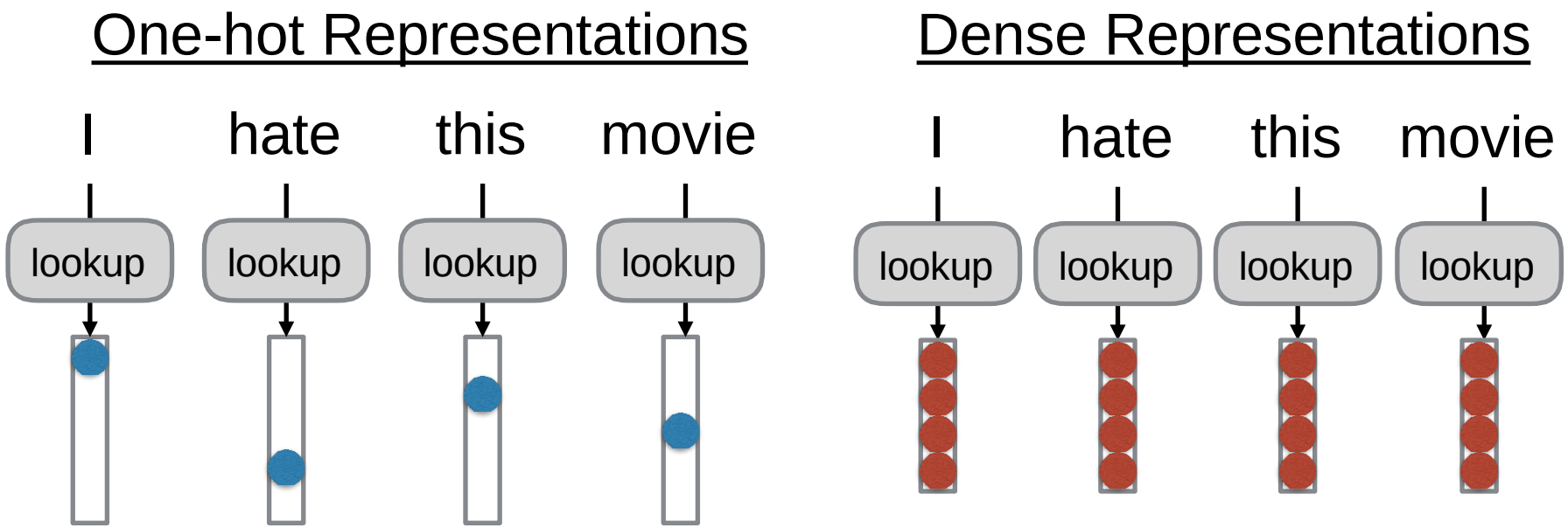


One hot vector

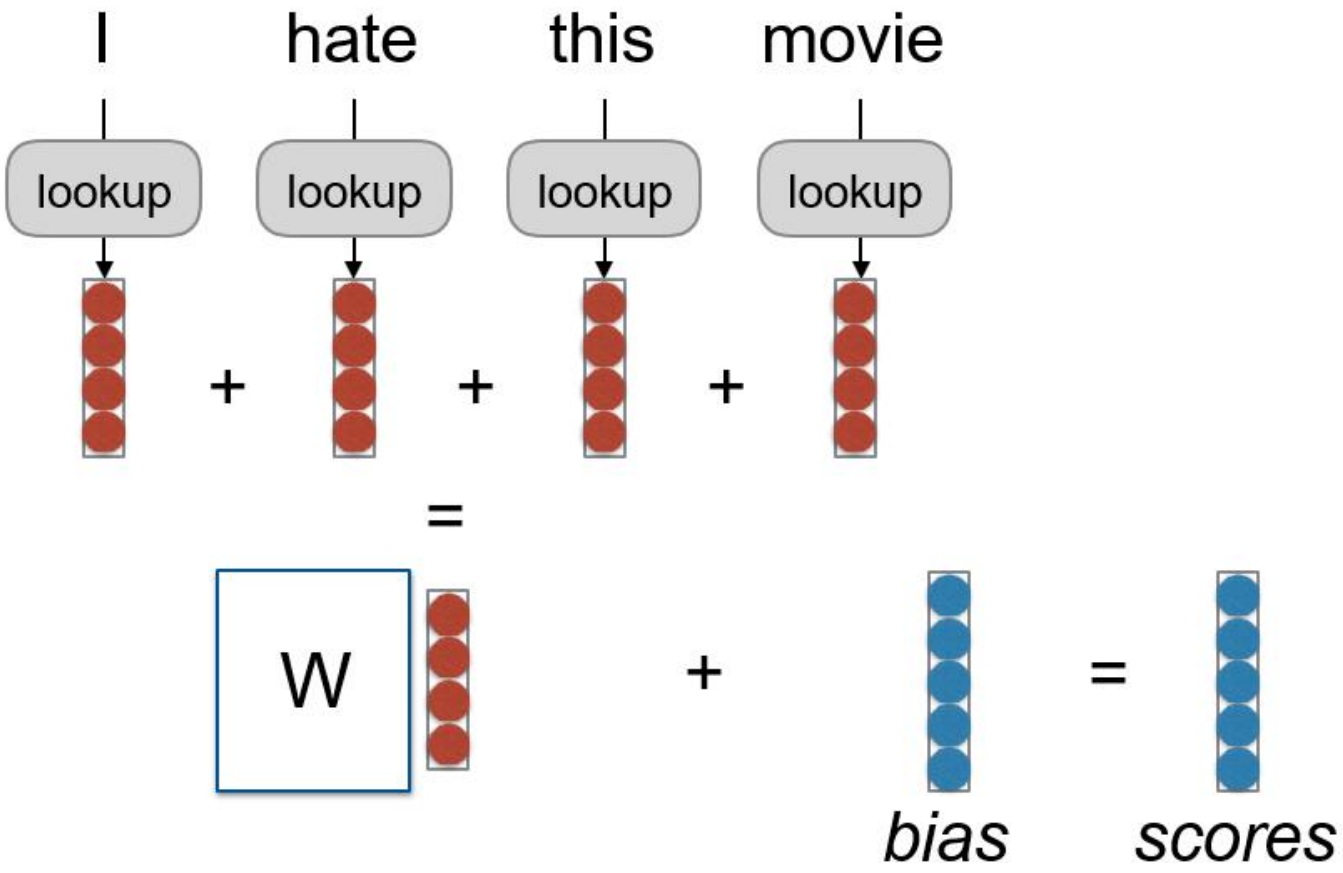
Features *f* are based on word identity, weights *w* learned

Word representation

- Previously we represented words with a *sparse* vector with a single "1" — a **one-hot** vector
- Continuous word embeddings look up a *dense* vector

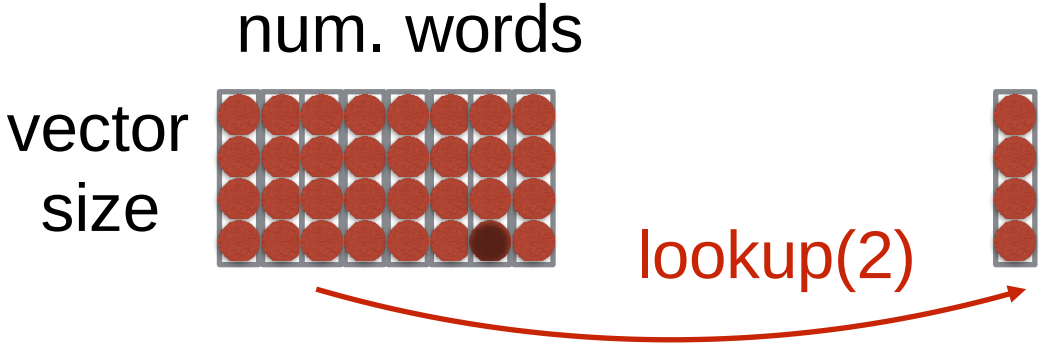


Continuous Bag of Words (CBOW)

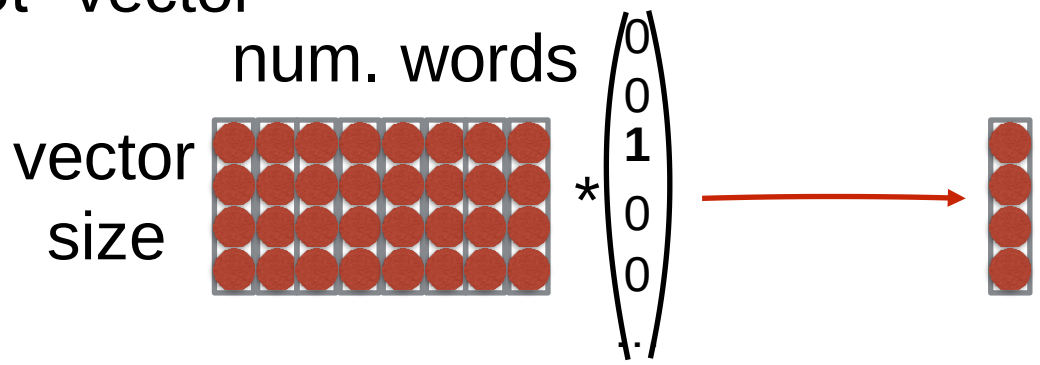


A Note: “Lookup”

- Lookup can be viewed as “grabbing” a single vector from a big matrix of word embeddings

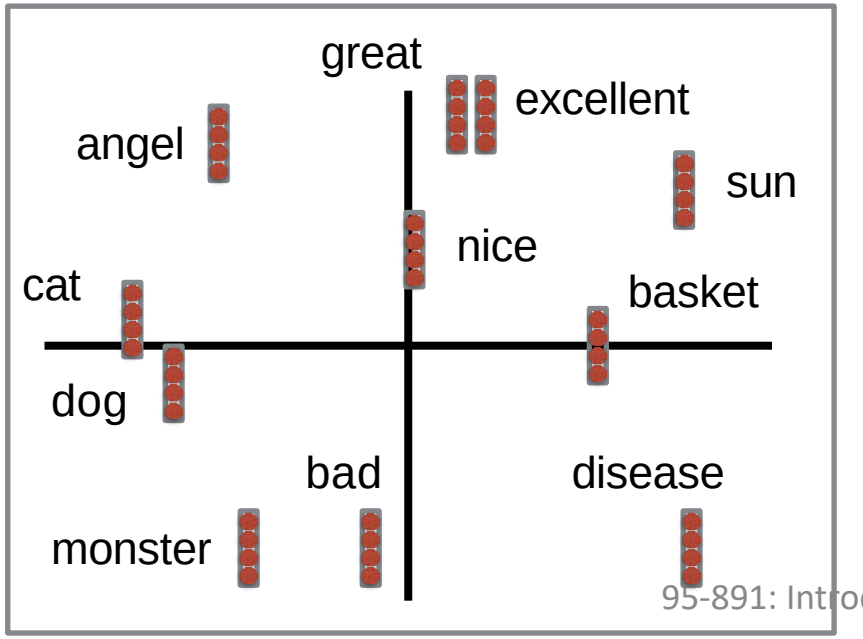


- Similarly, can be viewed as multiplying by a “one-hot” vector



What do Vectors Represent?

- No guarantees, but we hope that:
  - Words that are **similar** (syntactically, semantically, same language, etc.) are **close** in vector space
  - Each vector element is a **features** (e.g. is this an animate object? is this a positive word, etc.)



Shown in 2D, but in reality we use 512, 1024, etc.

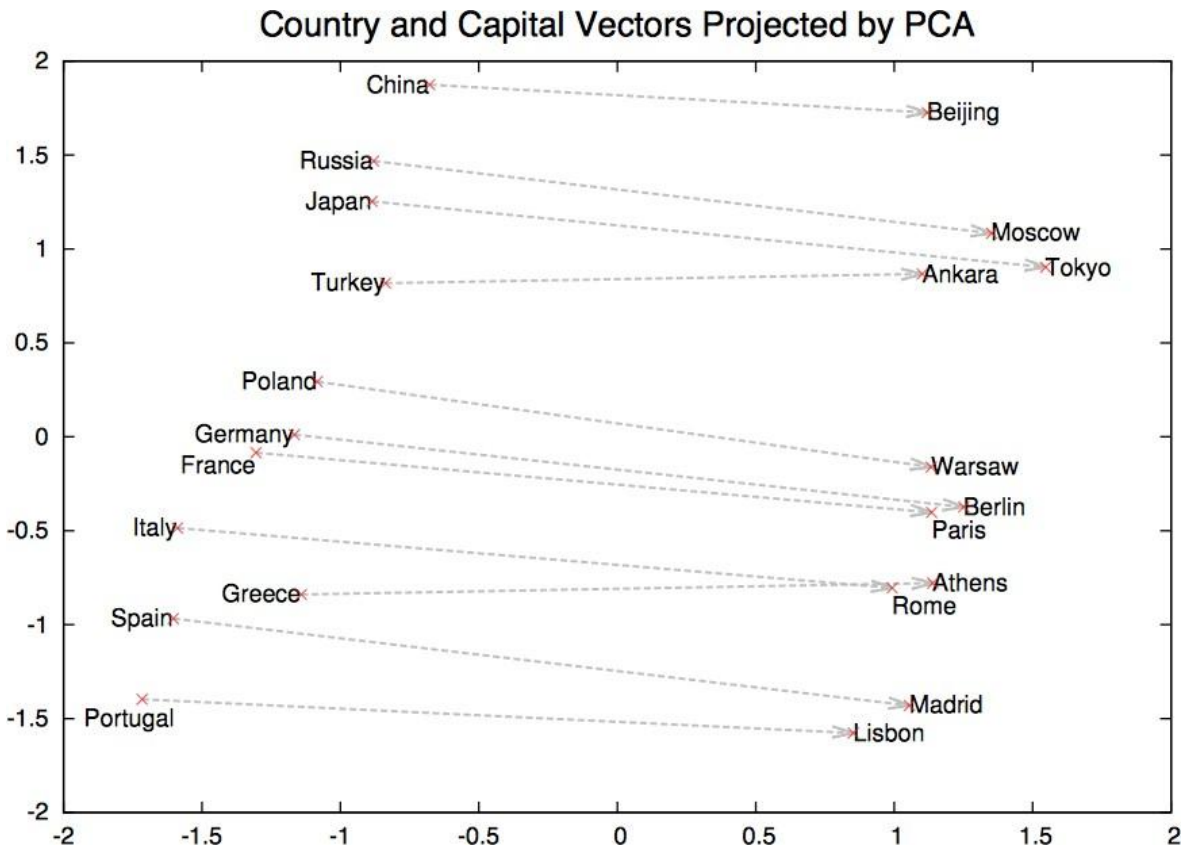
Vector space of word embeddings

- While learning these word representations, we are actually building a vector space in which all words reside with certain relationships between them
- Encodes both syntactic and semantic relationships
- This vector space allows for algebraic operations:

$$\text{Vec(king)} - \text{vec(man)} + \text{vec(woman)} \approx \text{vec(queen)}$$

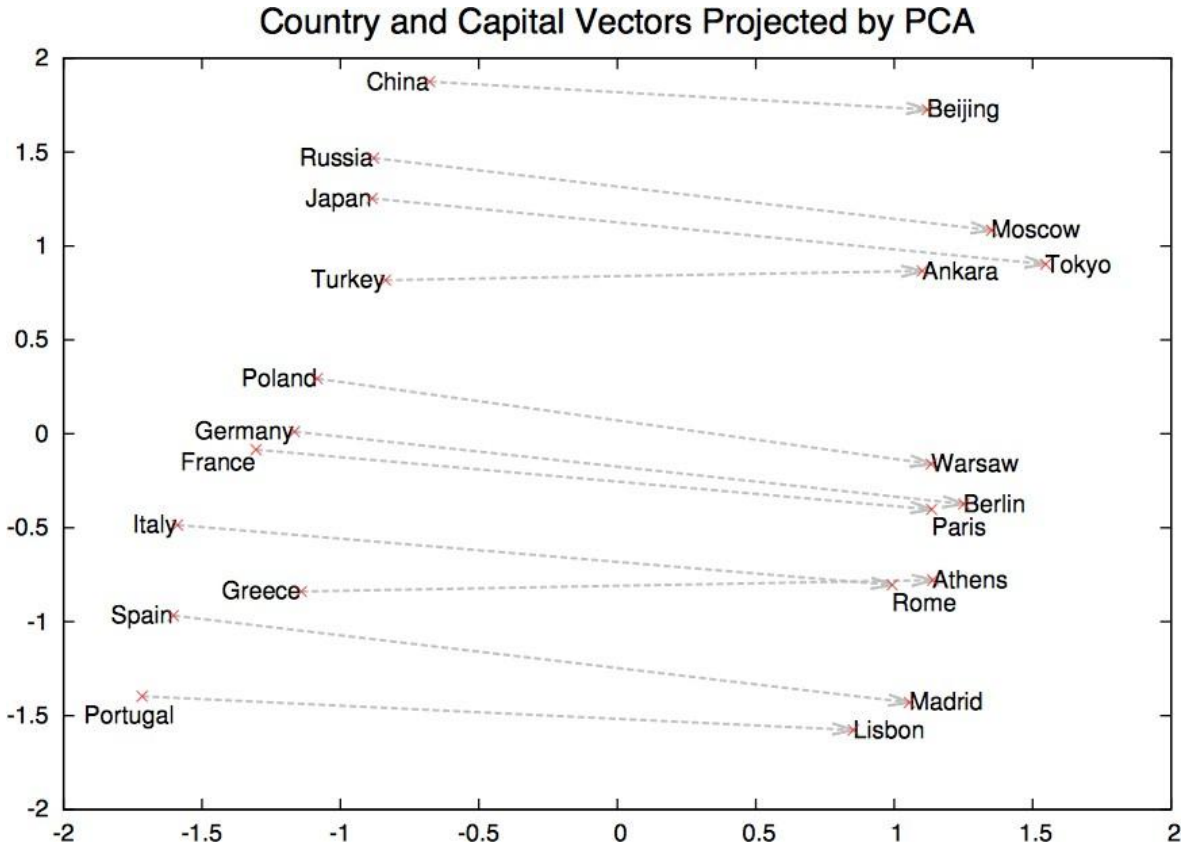
Vector space of word embeddings

- Reduce high-dimensional embeddings into 2/3D for visualization (e.g. Mikolov et al. 2013)



Vector space of word embeddings

- Reduce high-dimensional embeddings into 2/3D for visualization (e.g. Mikolov et al. 2013)



Do these work?
Issues of bias here

e.g. https://arxiv.org/abs/1607.06520
https://aclanthology.org/W14-1618.pdf

vec(programmer) – vec(man) +
vec(woman) ≈ vec(homemaker)

What is the meaning of “bardiwac”?

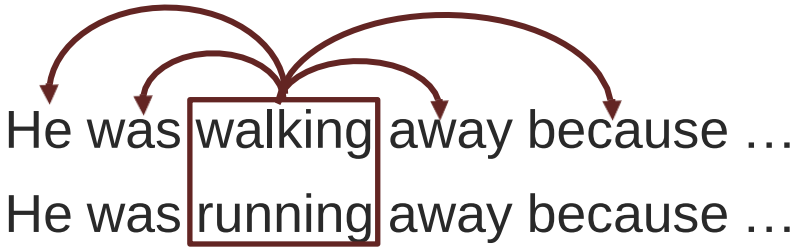
- He handed her a glass of **bardiwac**.
- Beef dishes are made to complement the **bardiwacs**.
- Nigel staggered to his feet, face flushed from too much **bardiwac**.
- Malbec, one of the lesser-known **bardiwac** grapes, responds well to Australia’s sunshine.
- I dined off bread and cheese and this excellent **bardiwac**.
- The drinks were delicious: blood-red **bardiwac** as well as light, sweet Rhenish.

=> **Bardiwac** is a heavy red alcoholic beverage made from grapes.

How to learn word representations

⭐ **Distribution hypothesis**: Approximate the word meaning by its surrounding words.

⭐ Words used in a similar context will lie close together



Word representation – Tokenization

- Word-level
  The companies are expanding => “The”, “companies”, “are”, “expanding”

- Char-level
  The companies are expanding => “T”, “h”, “e”, “c”, “o”, “m” .....

- Subword-level

Subword tokenization

- Split less common words into multiple **subword tokens**

the companies are expanding

↓

the **compan** **_ies** are **expand** **_ing**

Benefits:

- **Share parameters** between word variants, compound words
- Reduce parameter size, **save compute+memory**

GPT tokenizer: https://platform.openai.com/tokenizer

Language Modeling

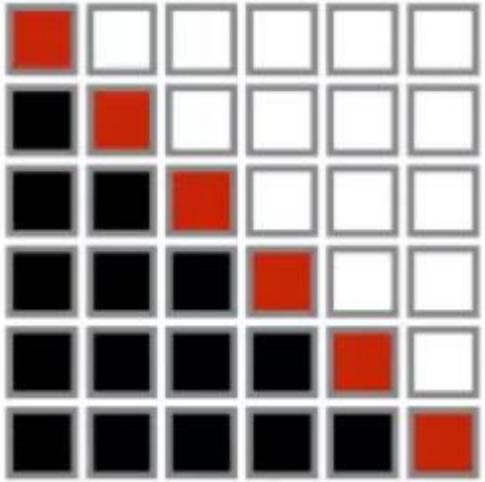
Predicting the next word

		$Prob.(next\_word \mid prefix)$
Santa Barbara has very nice ____	beach	0.5
	weather	0.4
	snow	0.01
Pittsburgh is a city of ____	bridges	0.6
	corn	0.02

Auto-regressive LM: Predicting the next word

$$P(X) = \prod_{i=1}^{I} P(x_i \mid x_1, \ldots, x_{i-1})$$

Next Token — Context



Chain rule:

$$\begin{aligned} P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \end{aligned}$$

What can we do with LMs?

- **Score** sentences:

  P(Jane went to the store .) → high
  P(store to Jane went the .) → low

  (same as calculating loss for training)

- **Generate** sentences:

$$\tilde{x} \sim P(X)$$

Example

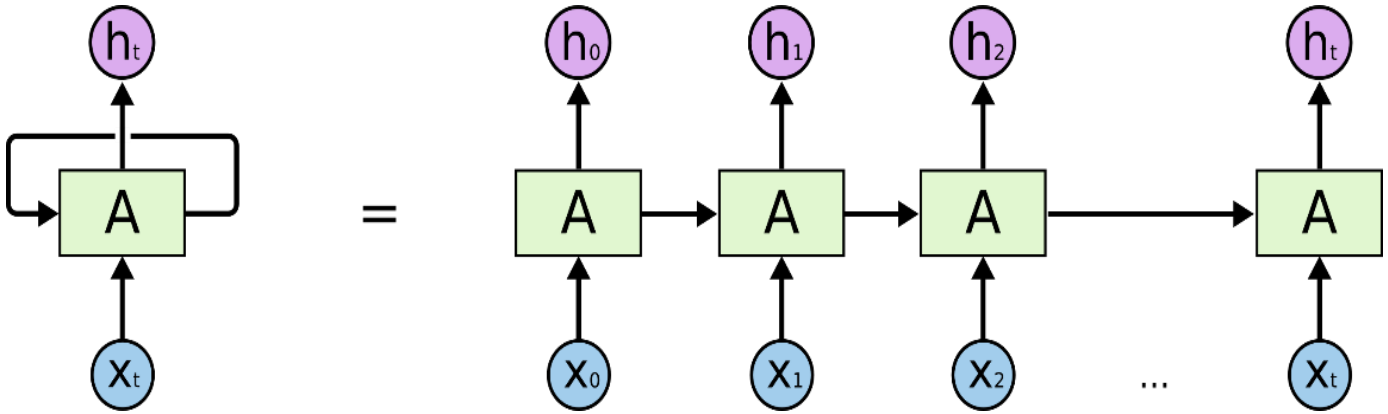
Previous words: “giving a"

a	3.0	-6.0	-0.2	-3.2
the	2.5	-5.1	-0.3	-2.9
talk	-0.2	0.2	1.0	1.0
gift	0.1	0.1	2.0	2.2
hat	1.2	0.5	-1.2	0.6
…	…	…	…	…
	b	$w_{1,a}$	$w_{2,giving}$	s
Words we’re predicting	How likely are they?	How likely are they given prev. word is “a”?	How likely are they given 2nd prev. word is “giving”?	Total score

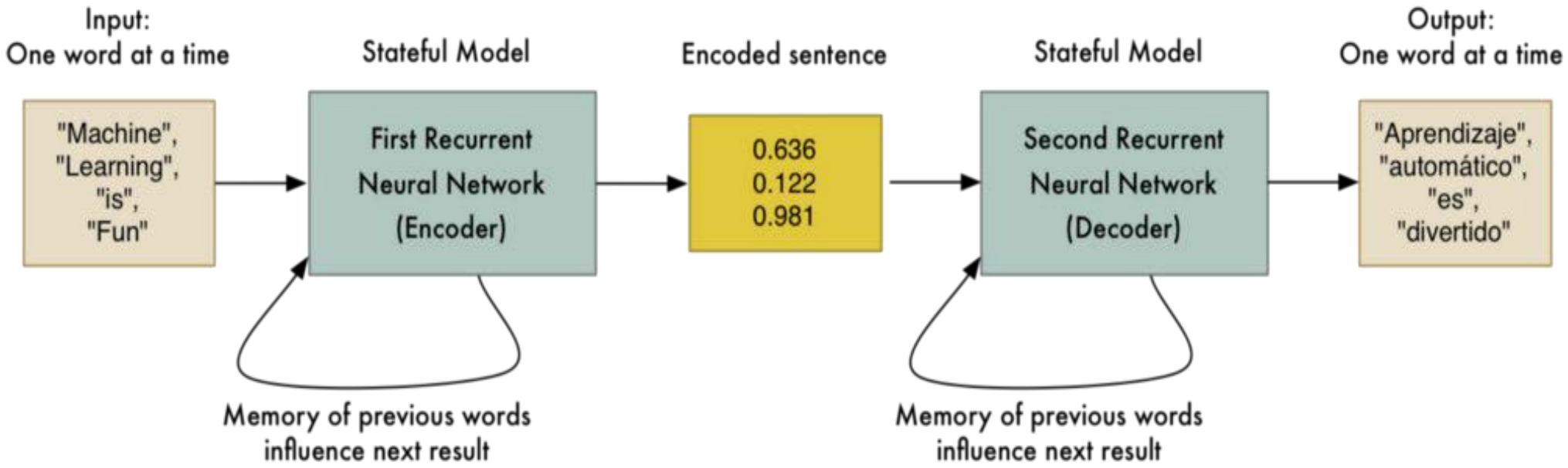
Recurrent Neural Network

Recurrent Neural Network (RNN) (Elman, 1990)

- (Session 8) Sequential data where time plays an important role.



Recurrent Neural Network (RNN)

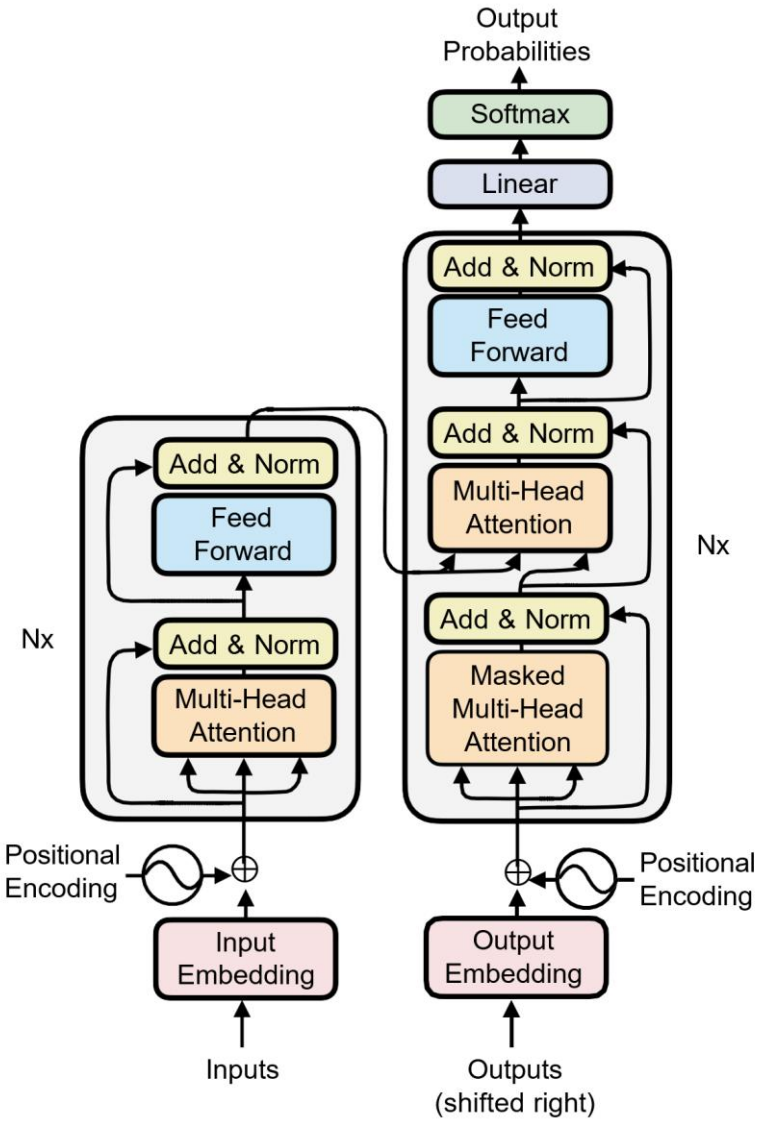


- Issues of RNN:
  - Slow computation for long sequences
  - Vanishing/exploding gradients
  - Difficulty in accessing information from long time ago

Transformer

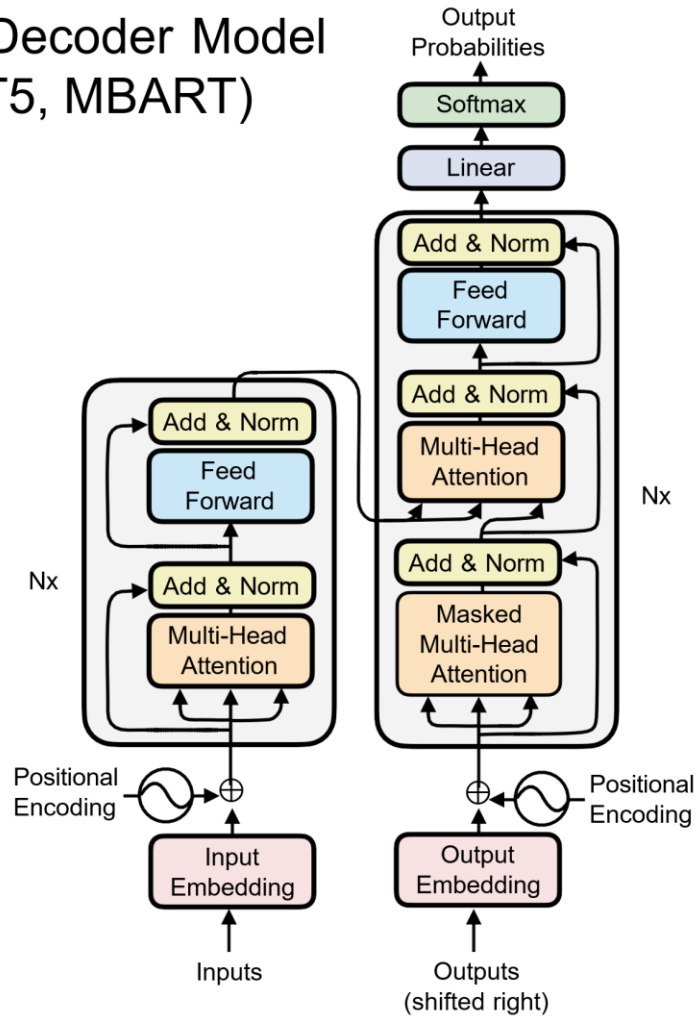
“Attention is All You Need” - Transformer

- A sequence-to-sequence model based entirely on attention
- SOTA performance on machine translation
- Fast: only matrix multiplications

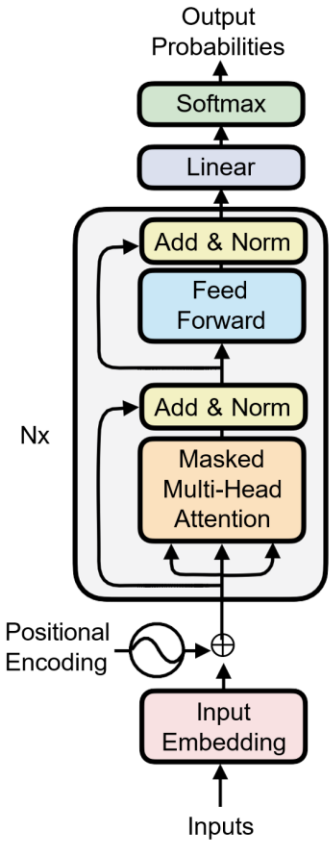


Two Types of Transformers

Encoder-Decoder Model
(e.g. T5, MBART)



Decoder Only Model
(e.g. GPT, LLaMa)



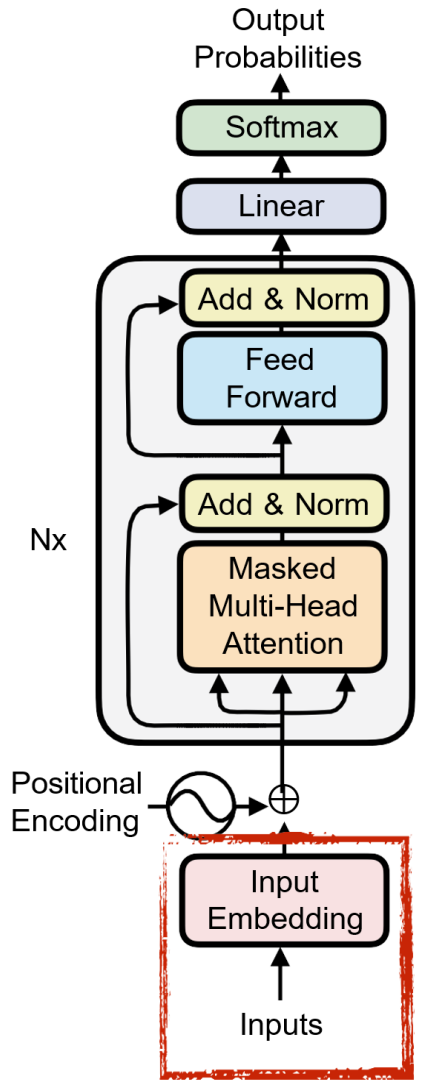
Inputs and embeddings

- Inputs: Generally, split using subwords

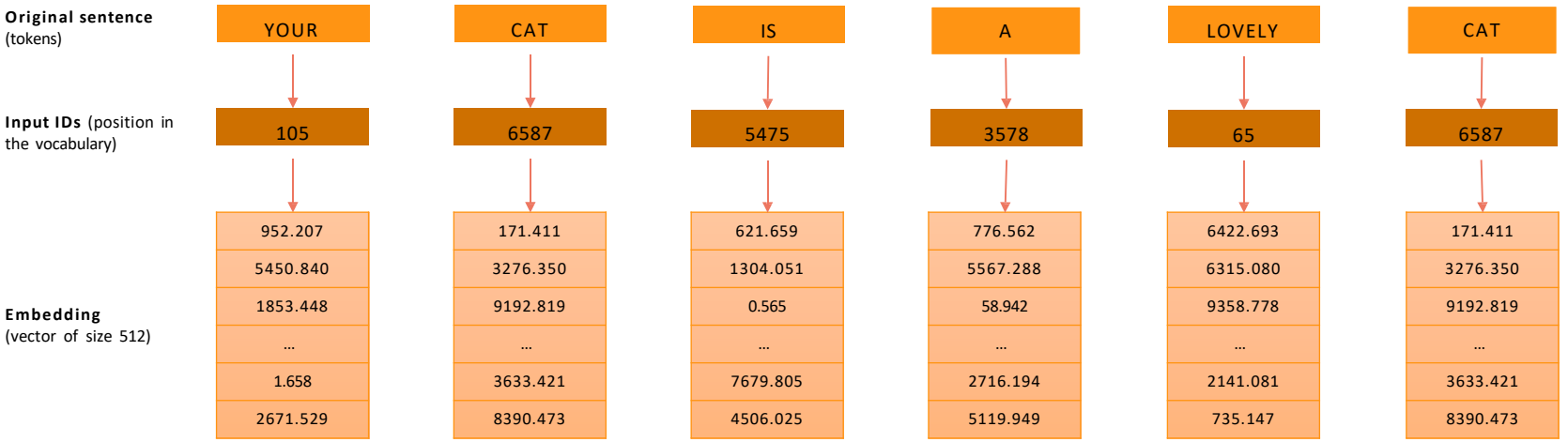
the books were improved

the book **_s** were **improv** **_ed**

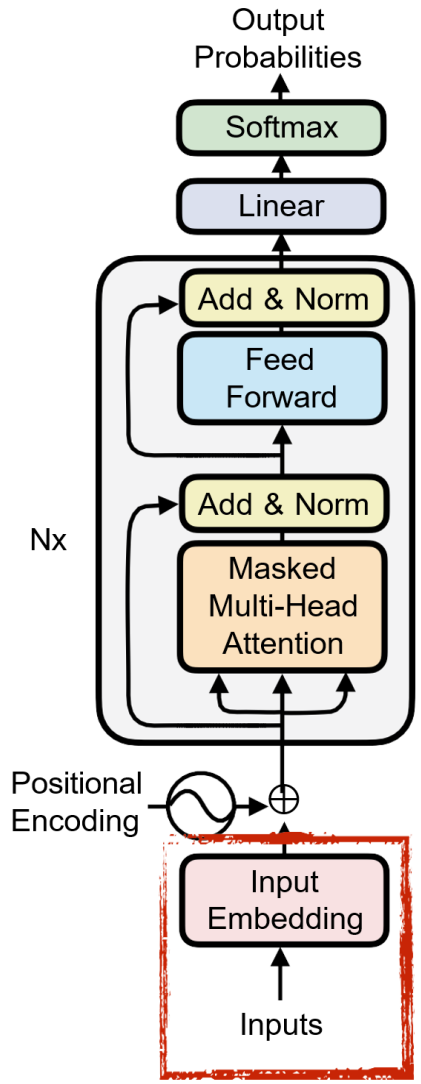
- Word Embedding: Looked up



Inputs and embeddings



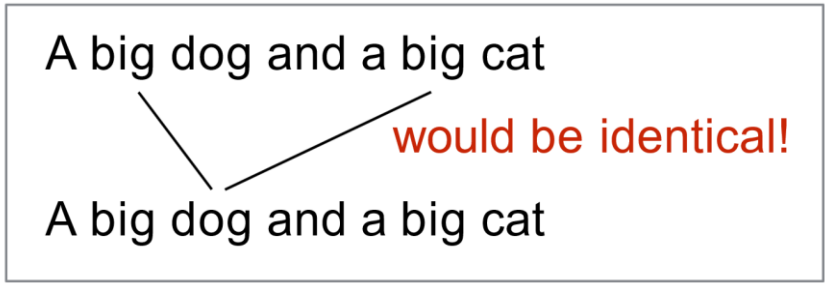
We define d_{model} = 512, which represents the size of the embedding vector of each word



Positional encoding

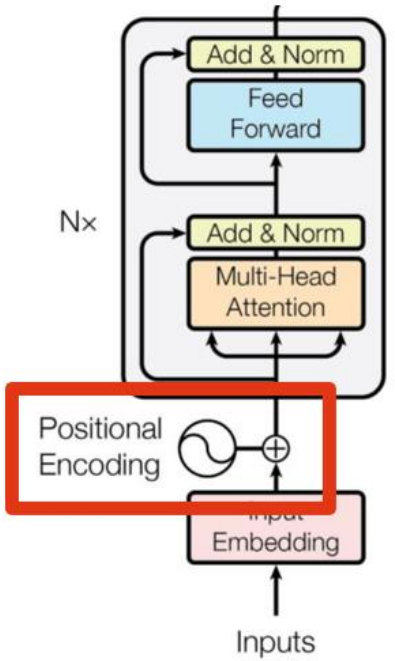
- The transformer model is *purely* attentional

- If embeddings were used, there would be no way to distinguish between *identical words*



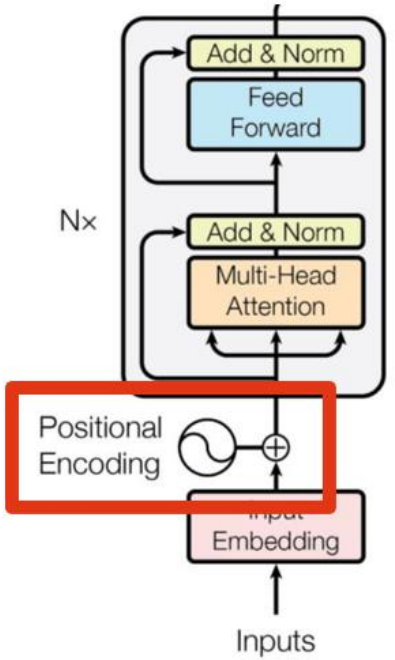
- Positional encodings add an embedding based on the word position

$W_{big} + W_{pos2}$ $W_{big} + W_{pos6}$



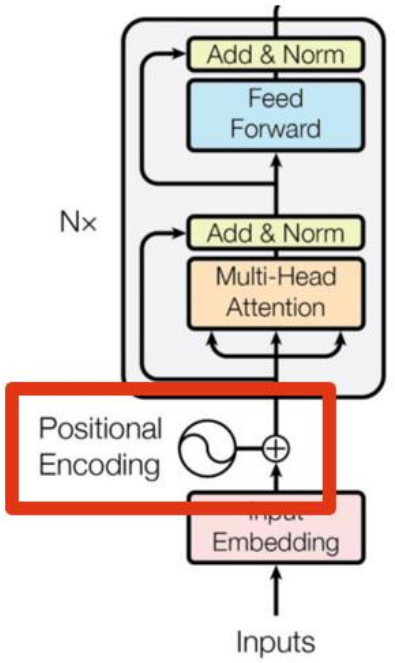
Positional encoding

	YOUR	CAT	IS	A	LOVELY	CAT
Original sentence						
Embedding (vector of size 512)	952.207 5450.840 1853.448 ... 1.658 2671.529	171.411 3276.350 9192.819 ... 3633.421 8390.473	621.659 1304.051 0.565 ... 7679.805 4506.025	776.562 5567.288 58.942 ... 2716.194 5119.949	6422.693 6315.080 9358.778 ... 2141.081 735.147	171.411 3276.350 9192.819 ... 3633.421 8390.473
	+	+	+	+	+	+
Position Embedding (vector of size 512). Only computed once and reused for every sentence during training and inference.	... ...	1664.068 8080.133 2620.399 ... 9386.405 3120.159	... ...	... ...	... ...	1281.458 7902.890 912.970 3821.102 1659.217 7018.620
	=	=	=	=	=	=
Encoder Input (vector of size 512)	... ...	1835.479 11356.483 11813.218 ... 13019.826 11510.632	... ...	... ...	... ...	1452.869 11179.24 10105.789 ... 5292.638 15409.093



Positional encoding

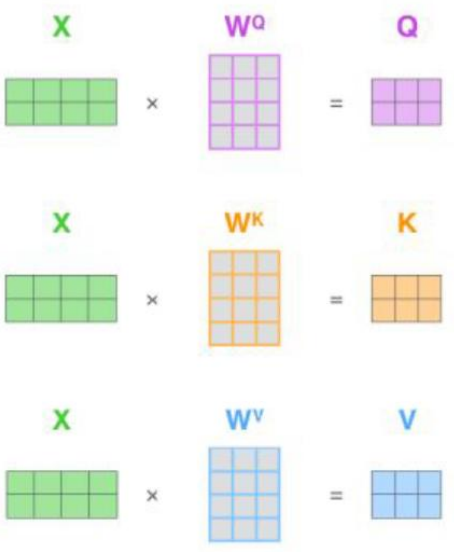
- **Absolute positional** encodings add an encoding to the input in hope that relative position will be captured. (Sinusoidal Encoding)

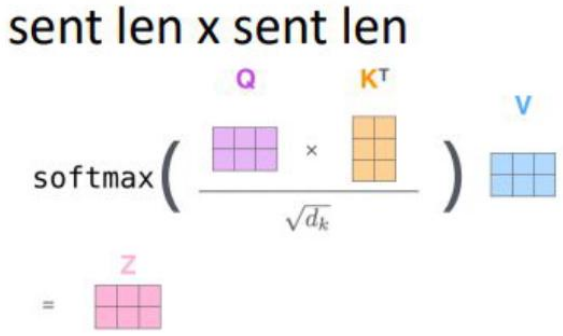


- **Relative positional** encodings explicitly encode relative position. (RoPE)

Attention

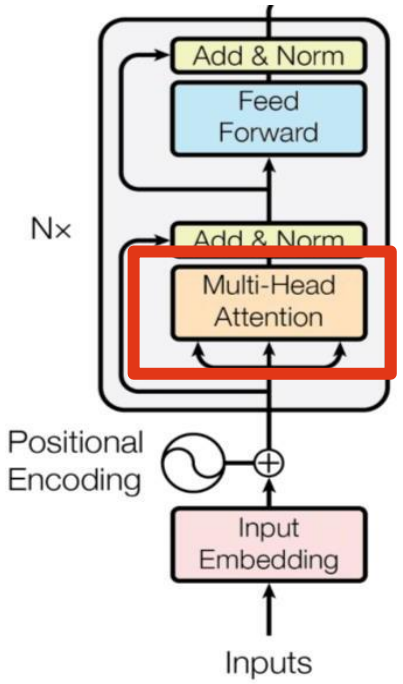
$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$





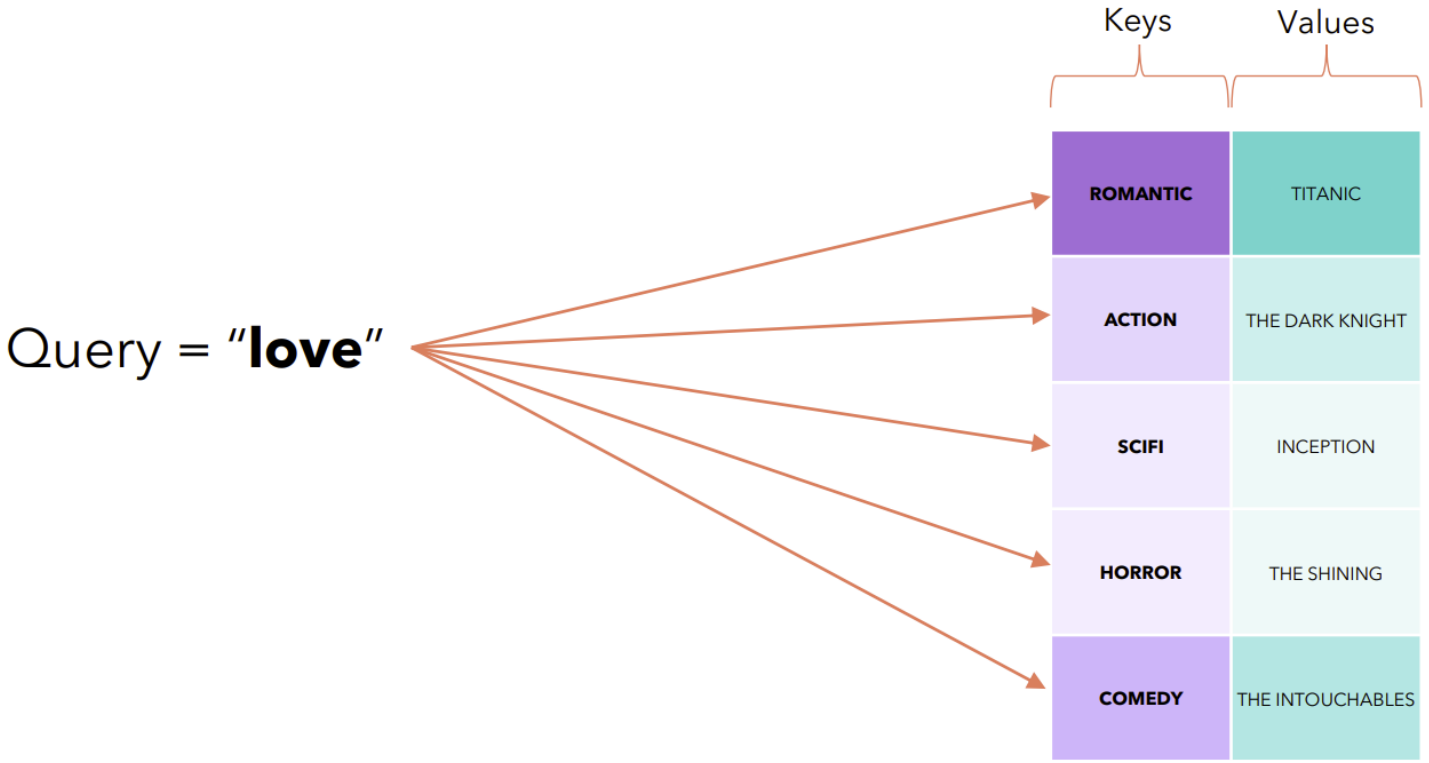
sent len x dim

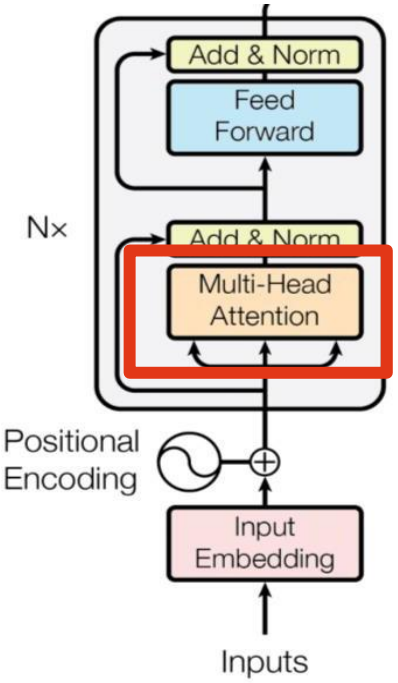
$$x'_i = \sum_{j=1}^{n} w_{ji} x_j$$



Why QKV?

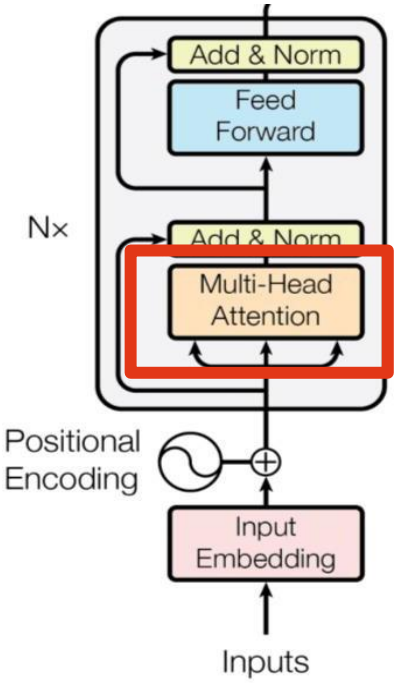
by giving each token a **query** (“what I’m looking for”), a **key** (“what I contain”), and a **value** (“what I contribute if I’m relevant”), we can systematically compute **how each token relates to every other token**—and then aggregate the information accordingly.





Types of Attention

- Self Attention
- Cross Attention
- Multi-Head Attention



Self-attention

Self-Attention allows the model to relate words to each other.

In this simple case we consider the sequence length **seq** = 6 and $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

The matrices **Q**, **K** and **V** are just the input sentence.

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{512}}\right) =$$

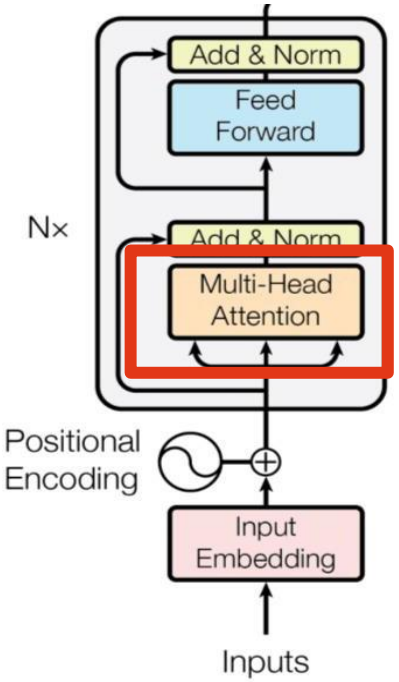
Q: (6, 512) × K^T: (512, 6)

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

* for simplicity I considered only one head, which makes $d_{model} = d_k$.

* all values are random.

(6, 6)



Cross-attention

self-attention is about understanding relationships within a single sequence, while cross-attention is about aligning or connecting information between two separate sequences.

In this simple case we consider the sequence length **seq** = 6 and $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

The matrices **Q**, **K** and **V** are just the input sentence.

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{512}}\right) =$$

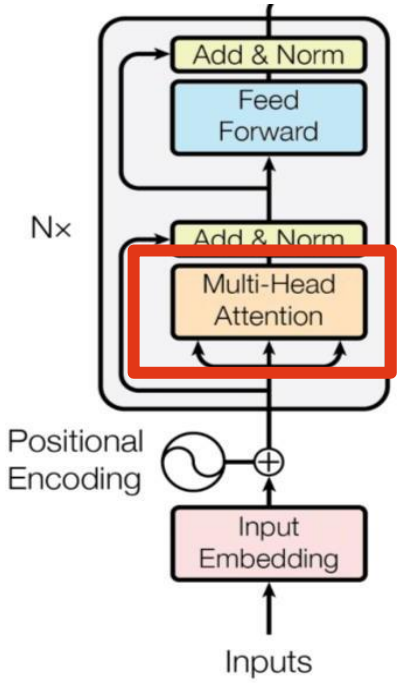
Q: (6, 512) × K^T: (512, 6)

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
你的	0.268	0.119	0.134	0.148	0.179	0.152	1
猫咪	0.124	0.278	0.201	0.128	0.154	0.115	1
是	0.147	0.132	0.262	0.097	0.218	0.145	1
一只	0.210	0.128	0.206	0.212	0.119	0.125	1
可爱的	0.146	0.158	0.152	0.143	0.227	0.174	1
猫咪	0.195	0.114	0.203	0.103	0.157	0.229	1

* for simplicity I considered only one head, which makes $d_{model} = d_k$.

* all values are random.

(6, 6)



Multi-head attention

Multi-head attention is used so that the model can capture *multiple* types of relationships or "focuses" **in parallel**.

I **run** a small business

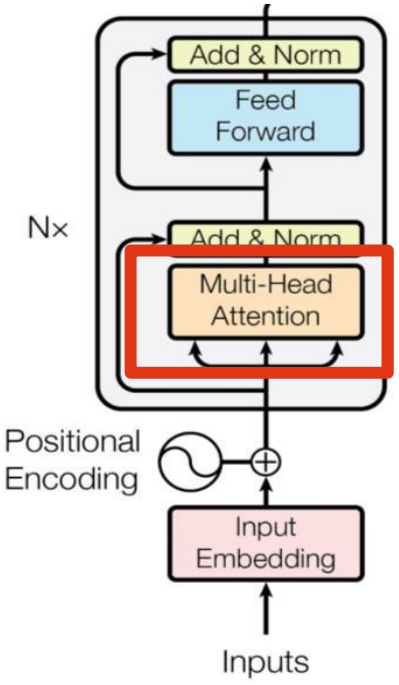
I **run** a mile in 10 minutes

The robber made a **run** for it

The stocking had a **run**

syntax
(nearby context)

semantics
(farther context)



Multi-head attention

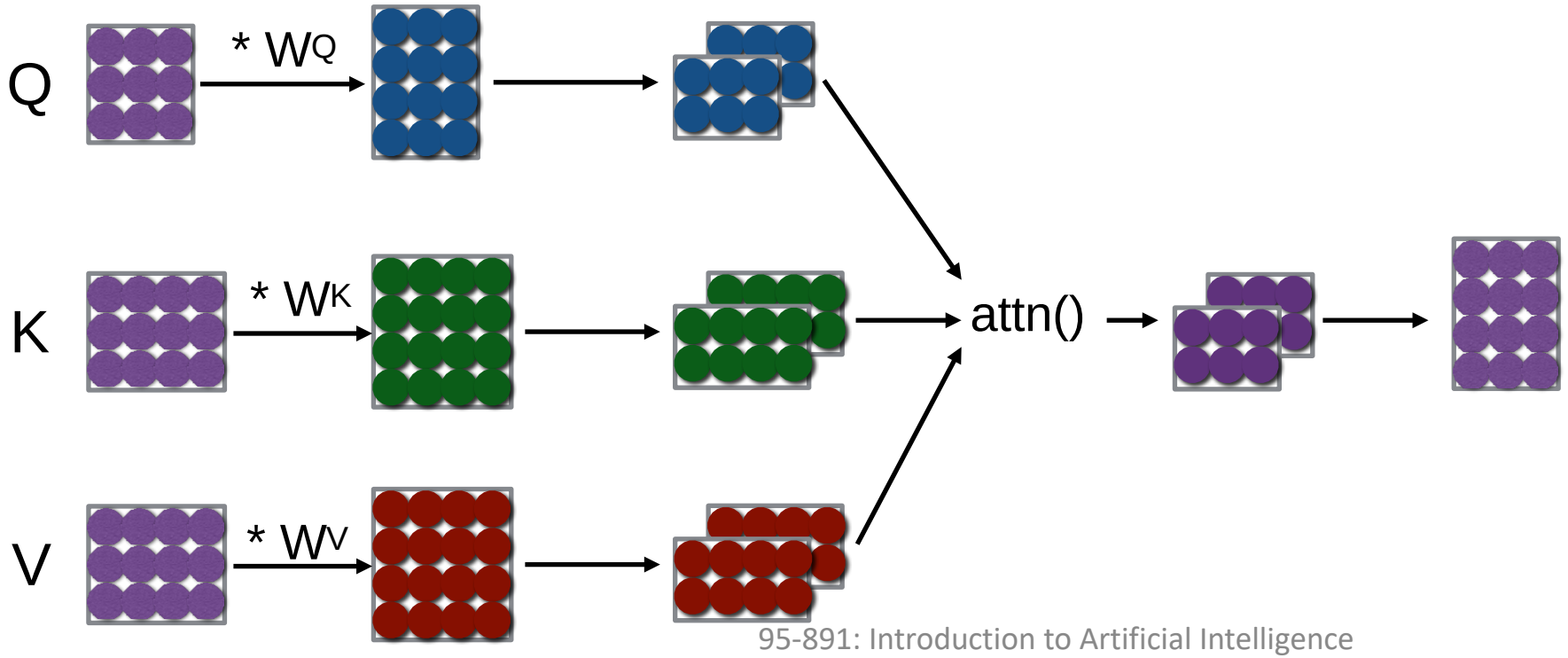
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

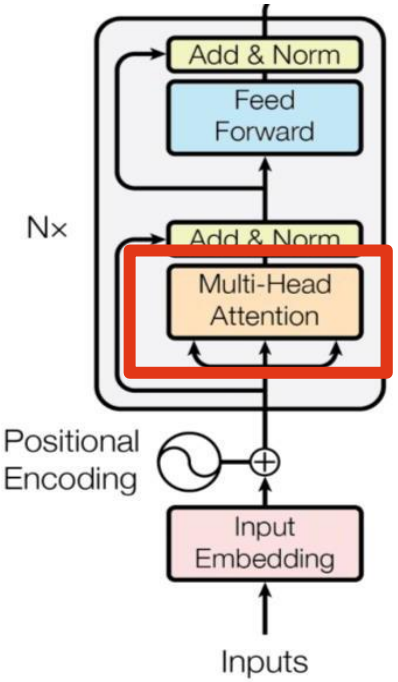
Multiply by weights

Split/rearrange to *n* attn inputs

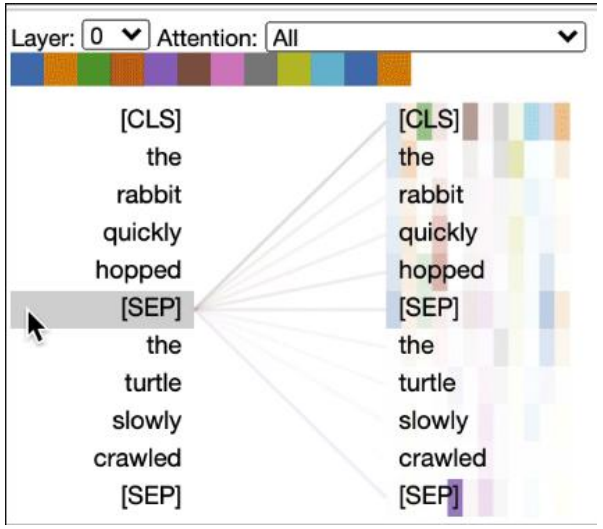
Run attn over each head

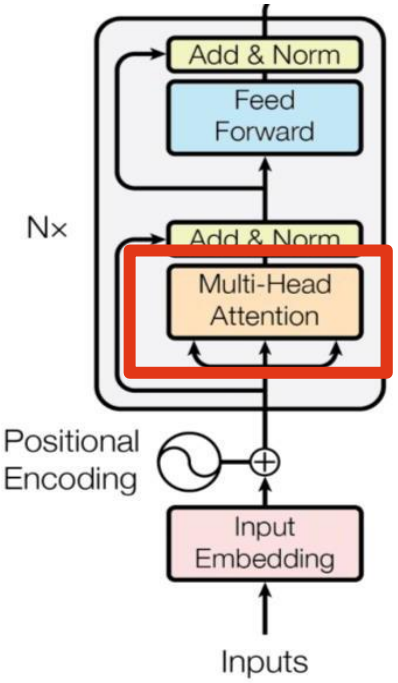
Concat and *W^O





Attention Visualization





https://github.com/jessevig/bertviz

Tomorrow's recitation

- Implementation in CV.

- Implementation in NLP with transformers.