Northeastern University
Khoury College of
Computer Sciences

Advanced Topics 3 - Attention

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

Wednesday | April 8th, 2026

Motivation

- Two main problems
  - Sequence Problems
  - Bottleneck Problems

Motivation

The Sequence Problem

- These problems involve variable-length sequences where order matters and elements relate to each other:
  - Machine translation: "The cat sat on the mat" → "Le chat s'est assis sur le tapis”
  - Text generation: predicting the next word in a document
  - Question answering: reading a paragraph and answering a question about it

Motivation

The Sequence Problem

- Why Not Just Use CNNs or MLPs?
  - MLPs require **fixed-size inputs** and treat each position independently - no way to handle variable length or capture positional relationships.
  - CNNs can handle sequences with 1D convolutions but have a **fixed receptive field** - a kernel of size k can only relate elements within k positions of each other.

Motivation

The Bottleneck Problem

- Imagine you are reading a very long book. To understand page 500, you need to remember a detail from page 1.

Motivation

The Bottleneck Problem

- Imagine you are reading a very long book. To understand page 500, you need to remember a detail from page 1.
- **Traditional Approach (Recurrent Neural Networks)**
    - You try to memorize the whole book in your head as you read.
    - By the time you reach page 500, the details from page 1 are blurry and forgotten.
    - This is the "Vanishing Gradient" or "Memory Bottleneck" problem.

Motivation

The Bottleneck Problem

- Imagine you are reading a very long book. To understand page 500, you need to remember a detail from page 1.
- **Attention Approach**
  - Instead of memorizing, you have a highlighter and an index.
  - Every time you read a new sentence, you “look back” at the entire book and highlight **only the parts relevant** to the current sentence.

Scaled Dot Product Attention

How a computer “looks” at other words

- Suppose you are in a library looking for information on “Climate Change”
  - **Query (Q):** This is what you are looking for (the search term in your head: "Climate Change”).
  - **Key (K):** This is the label on the spine of every book in the library ("Biology," "Weather," "Cooking," “History").
  - **Value (V):** This is the actual information inside the books.

Scaled Dot Product Attention

The Process

- You start with a one hot encoded vector for all words

Scaled Dot Product Attention

The Process

- You start with a one hot encoded vector for all words
- You then have **3 small neural networks** that “project” your one hot encoded vector into a d dimensional space - one for each of Query, Key and Value.

Scaled Dot Product Attention

The Process

- You start with a one hot encoded vector for all words
- You then have **3 small neural networks** that "project" your one hot encoded vector into a d dimensional space - one for each of Query, Key and Value.
- You compare your **Query** against all **Keys** to see how well they match.
- You get a "score" (similarity) for each book.

Scaled Dot Product Attention

The Process

- You start with a one hot encoded vector for all words
- You then have **3 small neural networks** that "project" your one hot encoded vector into a d dimensional space - one for each of Query, Key and Value.
- You compare your **Query** against all **Keys** to see how well they match.
- You get a "score" (similarity) for each book.
- You use those scores to decide how much of each book's **Value** to read.
  - If "Weather" matches highly, you read a lot of its content; if "Cooking" matches poorly, you ignore it.

Scaled Dot Product Attention

The Process

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

Scaled Dot Product Attention

The Process

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

QK^T : Measures similarity. High score = high relevance

Scaled Dot Product Attention

The Process

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

QK^T : Measures similarity. High score = high relevance

$\sqrt{d_k}$: As dimensions grow, dot products get huge, pushing the softmax into regions where gradients are tiny. Scaling keeps the math stable.

Scaled Dot Product Attention

The Process

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

QK^T : Measures similarity. High score = high relevance

$\sqrt{d_k}$: As dimensions grow, dot products get huge, pushing the softmax into regions where gradients are tiny. Scaling keeps the math stable.

Softmax: Converts scores into probabilities (0 to 1) that sum to 1. This tells us: “Give 80% attention to word A, 15% to word B, and 5% to word C.”

Scaled Dot Product Attention

The Process

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

QK^T : Measures similarity. High score = high relevance

$\sqrt{d_k}$: As dimensions grow, dot products get huge, pushing the softmax into regions where gradients are tiny. Scaling keeps the math stable.

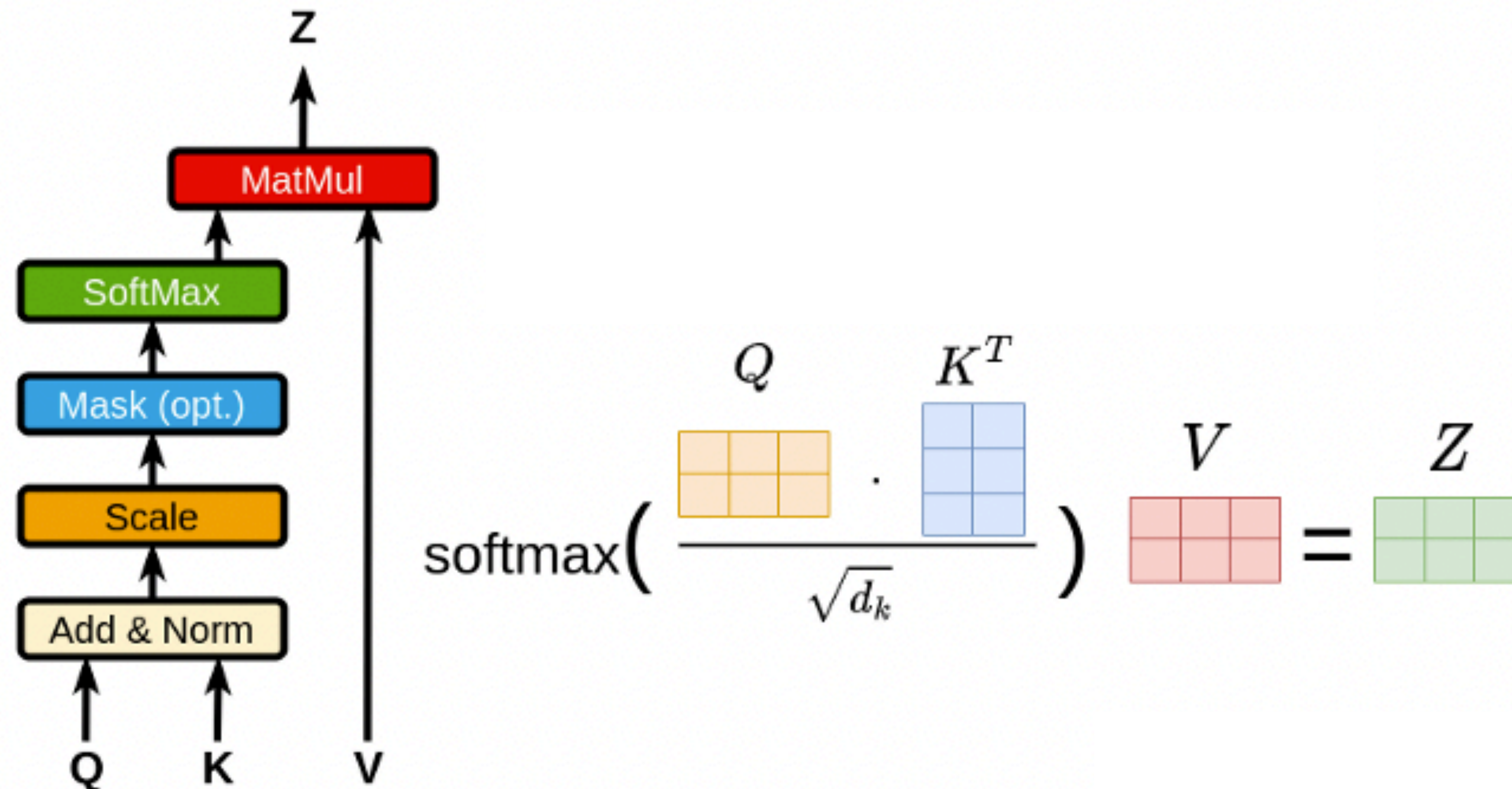
Softmax: Converts scores into probabilities (0 to 1) that sum to 1. This tells us: “Give 80% attention to word A, 15% to word B, and 5% to word C.”

$\times V$: We multiply the weights by the Values to get a single weighted sum vector.

Scaled Dot Product Attention

The Process

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



Scaled Dot Product Attention

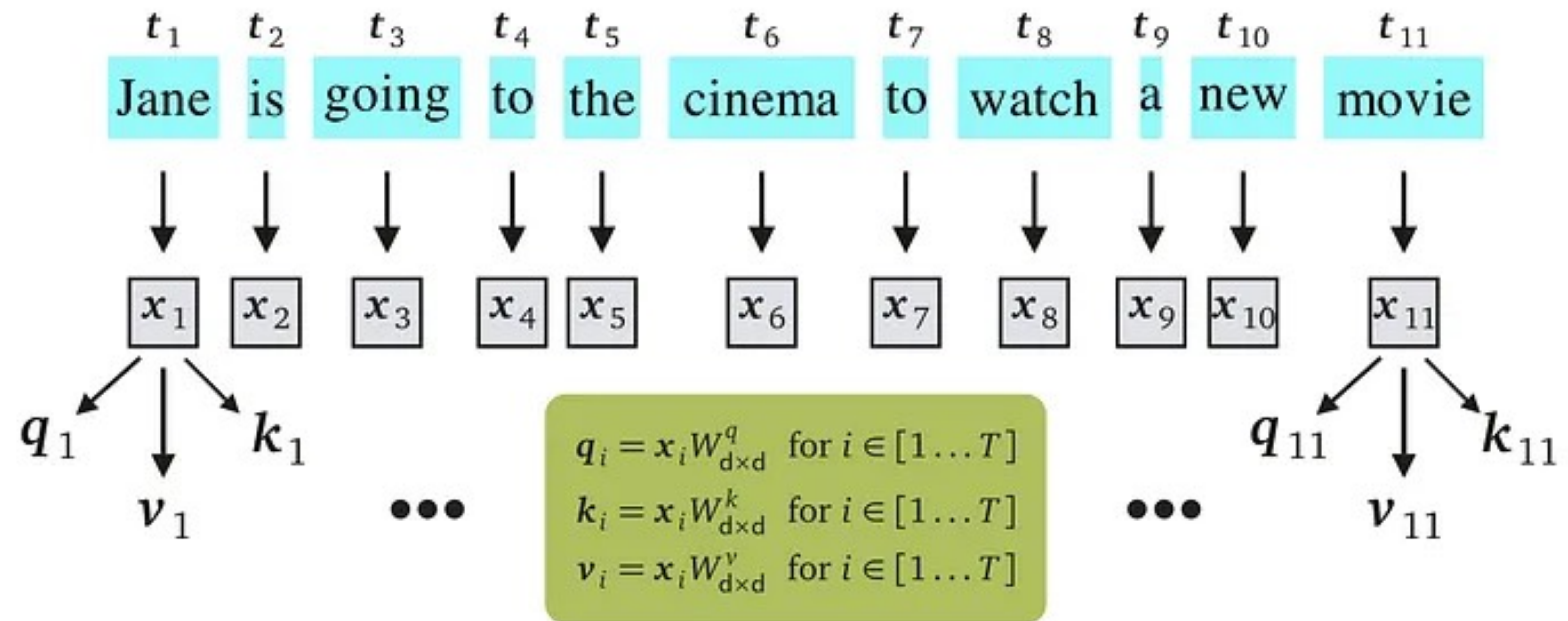
The Process

$q \in \mathbb{R}^{d_k}$ Query vector of size d_k $\longrightarrow q_{1\times d_k}$

$k \in \mathbb{R}^{d_k}$ Key vector of size d_k $\longrightarrow k_{1\times d_k}$

$v \in \mathbb{R}^{d_v}$ Value vector of size d_v $\longrightarrow v_{1\times d_v}$

In practice: $d_k = d_v = d$



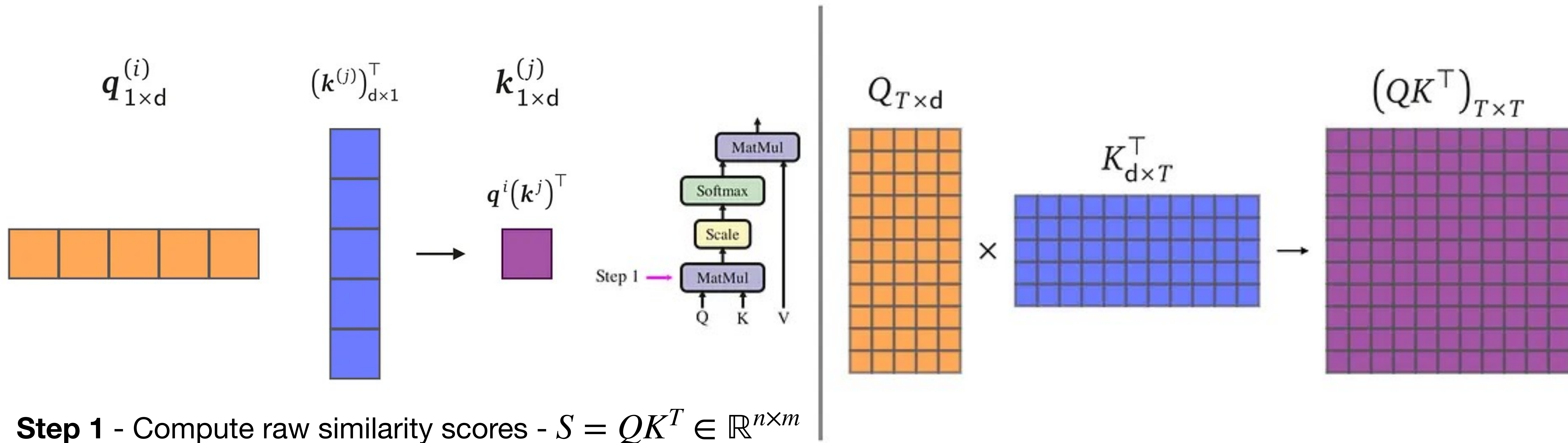
Step 1 - Compute raw similarity scores - $S = QK^T \in \mathbb{R}^{n\times m}$

The (i, j) entry of S is the dot product between query i and key j - a scalar measuring how similar they are.

Larger dot product = more similar.

Scaled Dot Product Attention

The Process



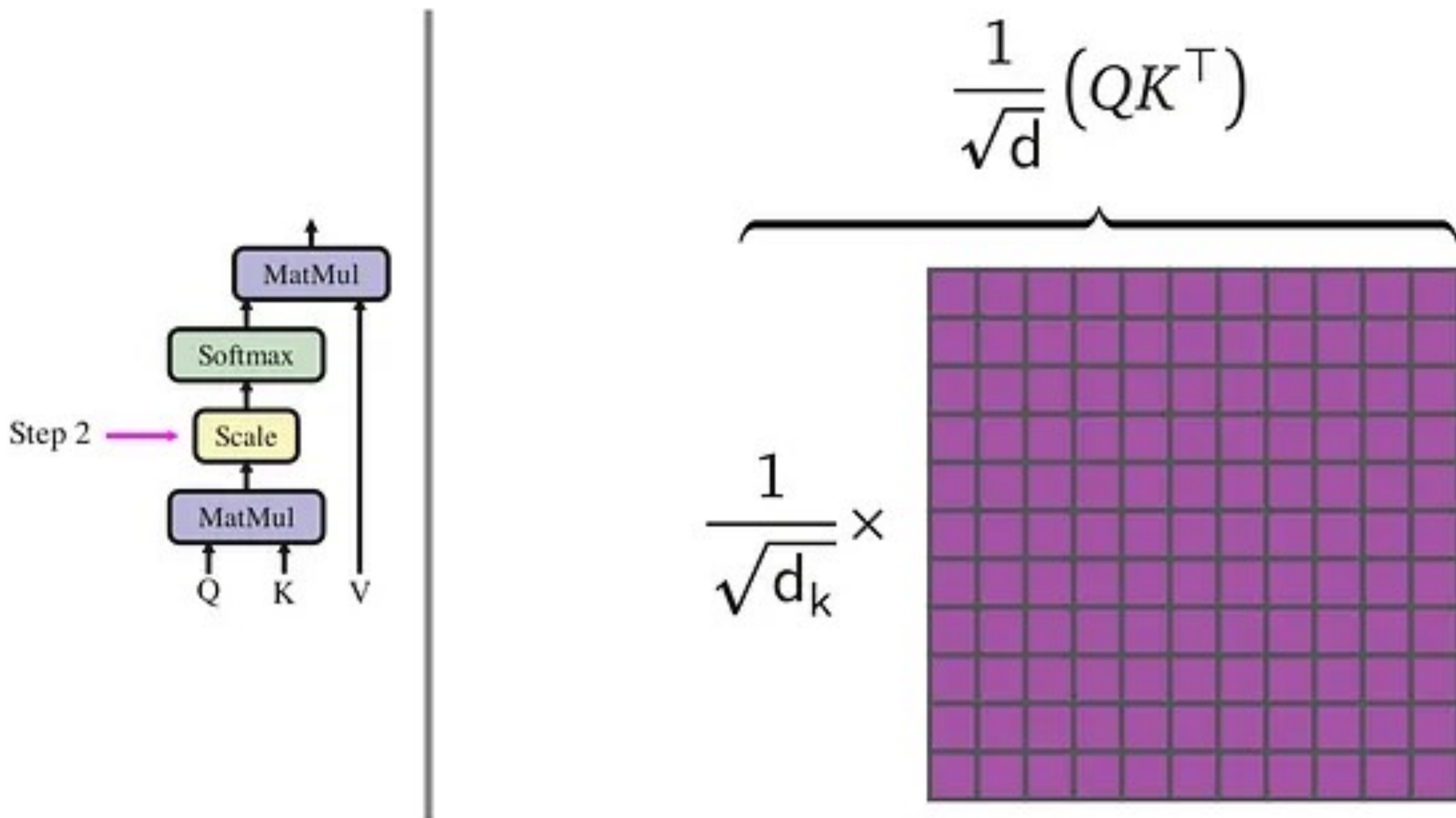
Step 1 - Compute raw similarity scores - $S = QK^T \in \mathbb{R}^{n\times m}$

The (i, j) entry of S is the dot product between query i and key j - a scalar measuring how similar they are.

Larger dot product = more similar.

Scaled Dot Product Attention

The Process



Attention

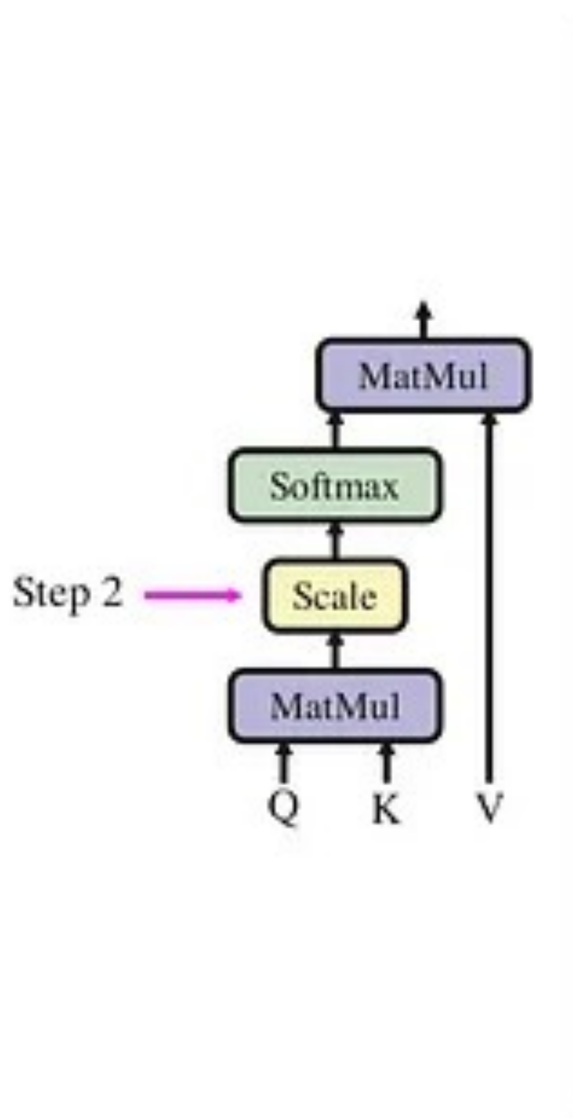
	a → $\vec{E}_1$ → $\vec{Q}_1$	fluffy → $\vec{E}_2$ → $\vec{Q}_2$	blue → $\vec{E}_3$ → $\vec{Q}_3$	creature → $\vec{E}_4$ → $\vec{Q}_4$	roamed → $\vec{E}_5$ → $\vec{Q}_5$	the → $\vec{E}_6$ → $\vec{Q}_6$	verdant → $\vec{E}_7$ → $\vec{Q}_7$	forest → $\vec{E}_8$ → $\vec{Q}_8$
a → $\vec{E}_1 \xrightarrow{W_k} \vec{K}_1$	$\vec{K}_1 \cdot \vec{Q}_1$	$\vec{K}_1 \cdot \vec{Q}_2$	$\vec{K}_1 \cdot \vec{Q}_3$	$\vec{K}_1 \cdot \vec{Q}_4$	$\vec{K}_1 \cdot \vec{Q}_5$	$\vec{K}_1 \cdot \vec{Q}_6$	$\vec{K}_1 \cdot \vec{Q}_7$	$\vec{K}_1 \cdot \vec{Q}_8$
fluffy → $\vec{E}_2 \xrightarrow{W_k} \vec{K}_2$	$\vec{K}_2 \cdot \vec{Q}_1$	$\vec{K}_2 \cdot \vec{Q}_2$	$\vec{K}_2 \cdot \vec{Q}_3$	$\vec{K}_2 \cdot \vec{Q}_4$	$\vec{K}_2 \cdot \vec{Q}_5$	$\vec{K}_2 \cdot \vec{Q}_6$	$\vec{K}_2 \cdot \vec{Q}_7$	$\vec{K}_2 \cdot \vec{Q}_8$
blue → $\vec{E}_3 \xrightarrow{W_k} \vec{K}_3$	$\vec{K}_3 \cdot \vec{Q}_1$	$\vec{K}_3 \cdot \vec{Q}_2$	$\vec{K}_3 \cdot \vec{Q}_3$	$\vec{K}_3 \cdot \vec{Q}_4$	$\vec{K}_3 \cdot \vec{Q}_5$	$\vec{K}_3 \cdot \vec{Q}_6$	$\vec{K}_3 \cdot \vec{Q}_7$	$\vec{K}_3 \cdot \vec{Q}_8$
creature → $\vec{E}_4 \xrightarrow{W_k} \vec{K}_4$	$\vec{K}_4 \cdot \vec{Q}_1$	$\vec{K}_4 \cdot \vec{Q}_2$	$\vec{K}_4 \cdot \vec{Q}_3$	$\vec{K}_4 \cdot \vec{Q}_4$	$\vec{K}_4 \cdot \vec{Q}_5$	$\vec{K}_4 \cdot \vec{Q}_6$	$\vec{K}_4 \cdot \vec{Q}_7$	$\vec{K}_4 \cdot \vec{Q}_8$
roamed → $\vec{E}_5 \xrightarrow{W_k} \vec{K}_5$	$\vec{K}_5 \cdot \vec{Q}_1$	$\vec{K}_5 \cdot \vec{Q}_2$	$\vec{K}_5 \cdot \vec{Q}_3$	$\vec{K}_5 \cdot \vec{Q}_4$	$\vec{K}_5 \cdot \vec{Q}_5$	$\vec{K}_5 \cdot \vec{Q}_6$	$\vec{K}_5 \cdot \vec{Q}_7$	$\vec{K}_5 \cdot \vec{Q}_8$
the → $\vec{E}_6 \xrightarrow{W_k} \vec{K}_6$	$\vec{K}_6 \cdot \vec{Q}_1$	$\vec{K}_6 \cdot \vec{Q}_2$	$\vec{K}_6 \cdot \vec{Q}_3$	$\vec{K}_6 \cdot \vec{Q}_4$	$\vec{K}_6 \cdot \vec{Q}_5$	$\vec{K}_6 \cdot \vec{Q}_6$	$\vec{K}_6 \cdot \vec{Q}_7$	$\vec{K}_6 \cdot \vec{Q}_8$
verdant → $\vec{E}_7 \xrightarrow{W_k} \vec{K}_7$	$\vec{K}_7 \cdot \vec{Q}_1$	$\vec{K}_7 \cdot \vec{Q}_2$	$\vec{K}_7 \cdot \vec{Q}_3$	$\vec{K}_7 \cdot \vec{Q}_4$	$\vec{K}_7 \cdot \vec{Q}_5$	$\vec{K}_7 \cdot \vec{Q}_6$	$\vec{K}_7 \cdot \vec{Q}_7$	$\vec{K}_7 \cdot \vec{Q}_8$
forest → $\vec{E}_8 \xrightarrow{W_k} \vec{K}_8$	$\vec{K}_8 \cdot \vec{Q}_1$	$\vec{K}_8 \cdot \vec{Q}_2$	$\vec{K}_8 \cdot \vec{Q}_3$	$\vec{K}_8 \cdot \vec{Q}_4$	$\vec{K}_8 \cdot \vec{Q}_5$	$\vec{K}_8 \cdot \vec{Q}_6$	$\vec{K}_8 \cdot \vec{Q}_7$	$\vec{K}_8 \cdot \vec{Q}_8$

Each column: word → $\vec{E}_i \xrightarrow{W_Q} \vec{Q}_i$

Step 2 - Scale by $\sqrt{d_k}$

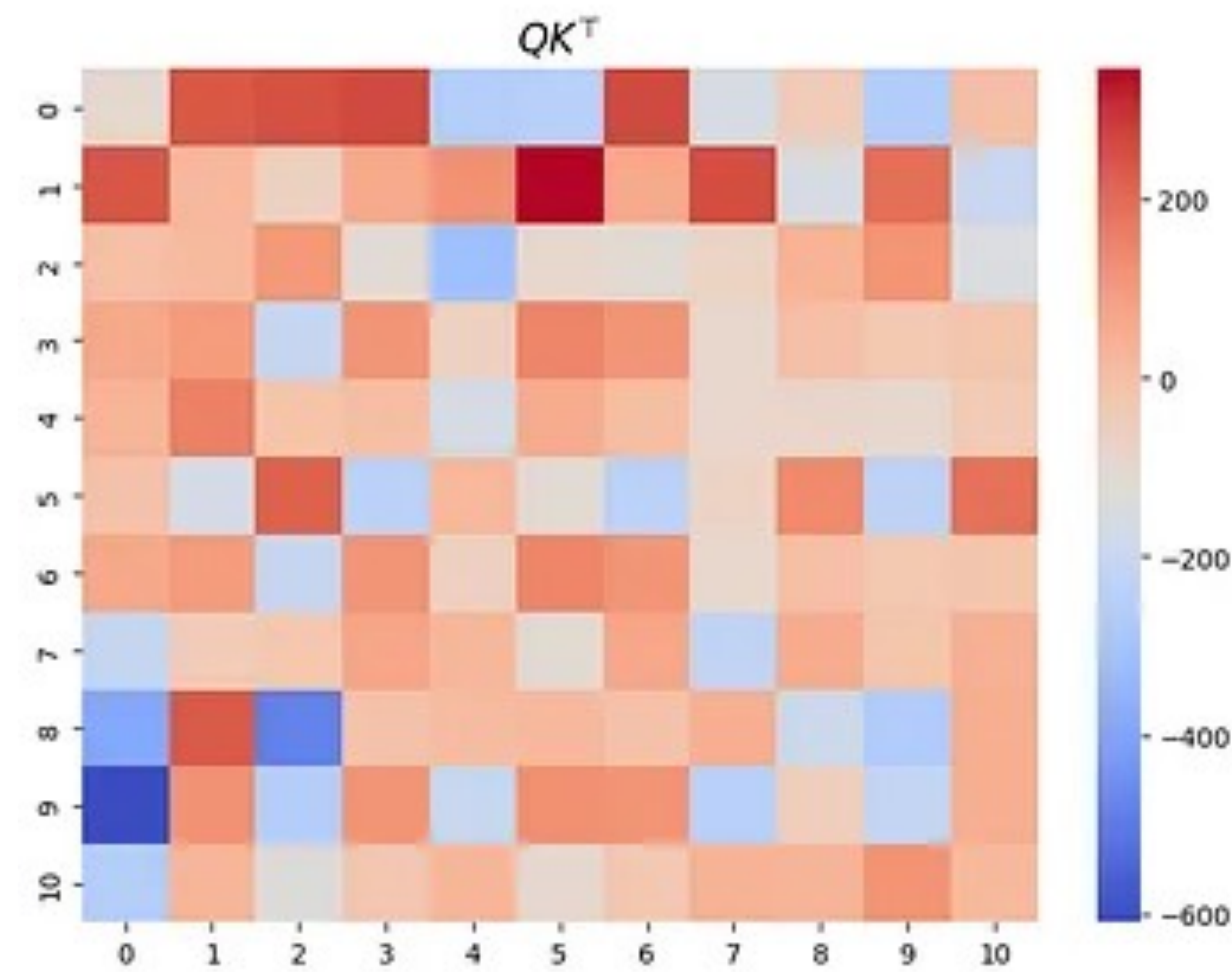
Scaled Dot Product Attention

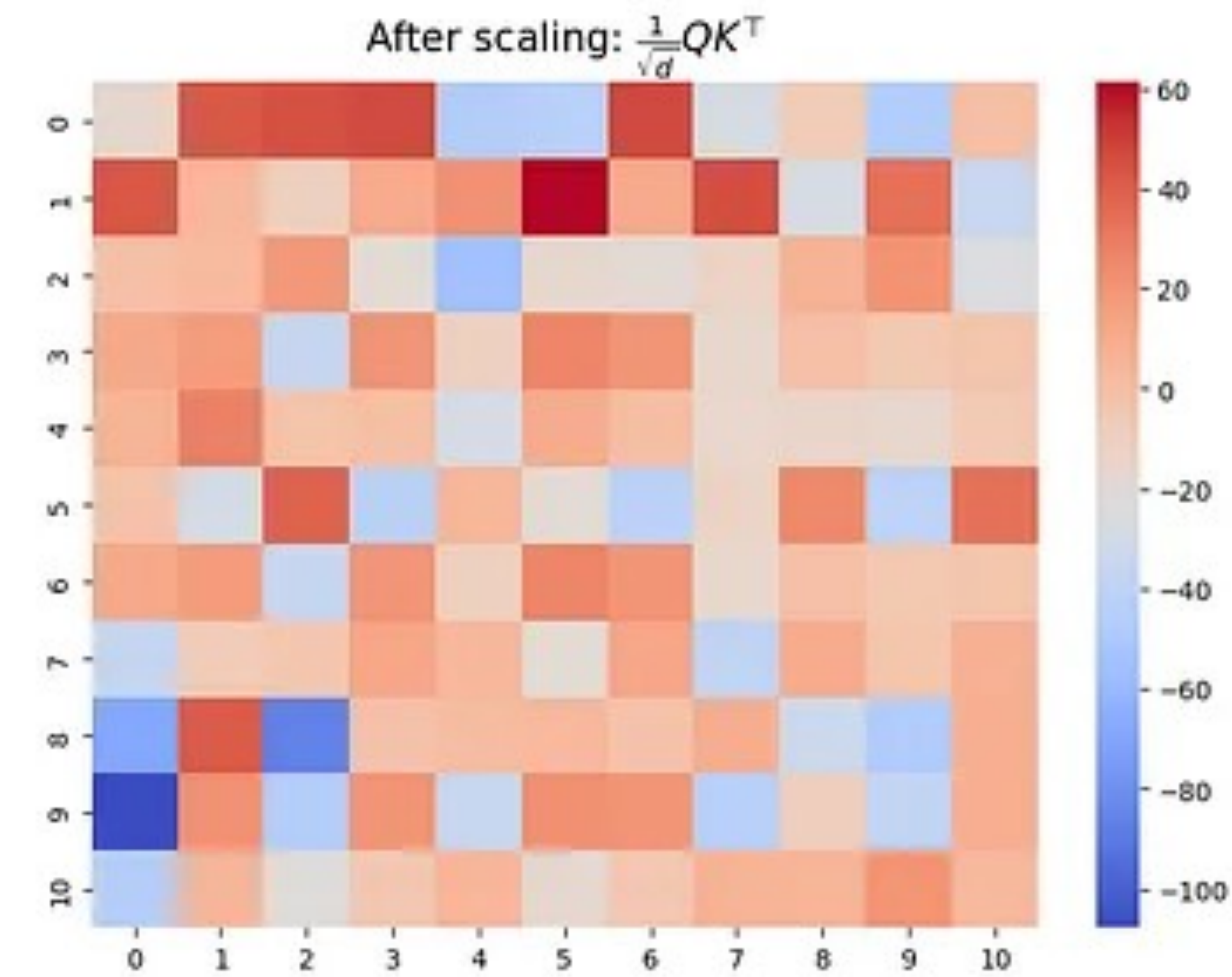
The Process



$$\frac{1}{\sqrt{d}}(QK^\top)$$

$$\frac{1}{\sqrt{d_k}} \times$$

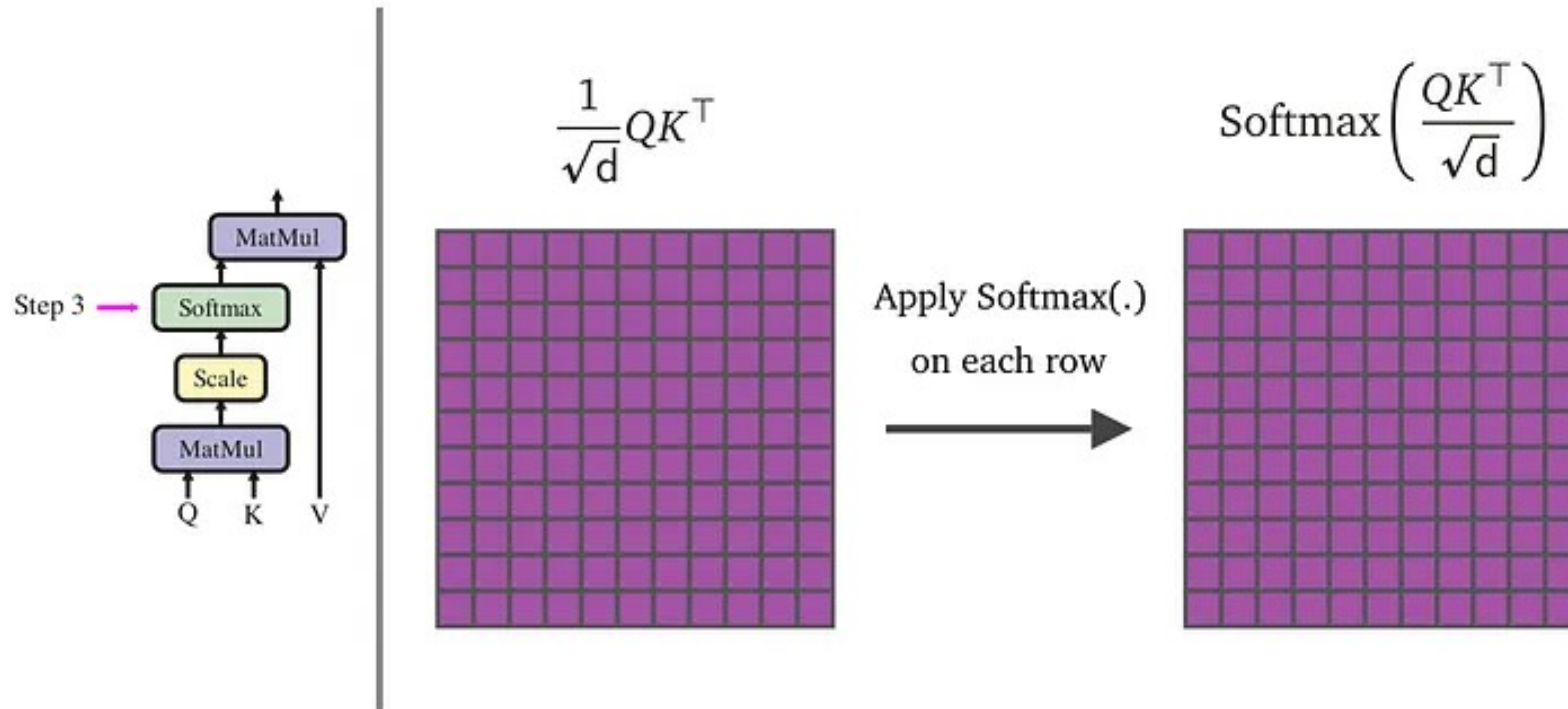




Step 2 - Scale by $\sqrt{d_k}$

Scaled Dot Product Attention

The Process



Step 3 - Softmax over keys (per query)

Applied **row-wise** - each query gets its own distribution over all keys.

This gives attention weights - non-negative, summing to 1 across the **m** keys for each query.

Scaled Dot Product Attention

The Process

MatMul

Softmax

Step 2 → Scale

MatMul

Q K V

$$\frac{1}{\sqrt{d}}(QK^{\top})$$

$$\frac{1}{\sqrt{d_k}} \times$$

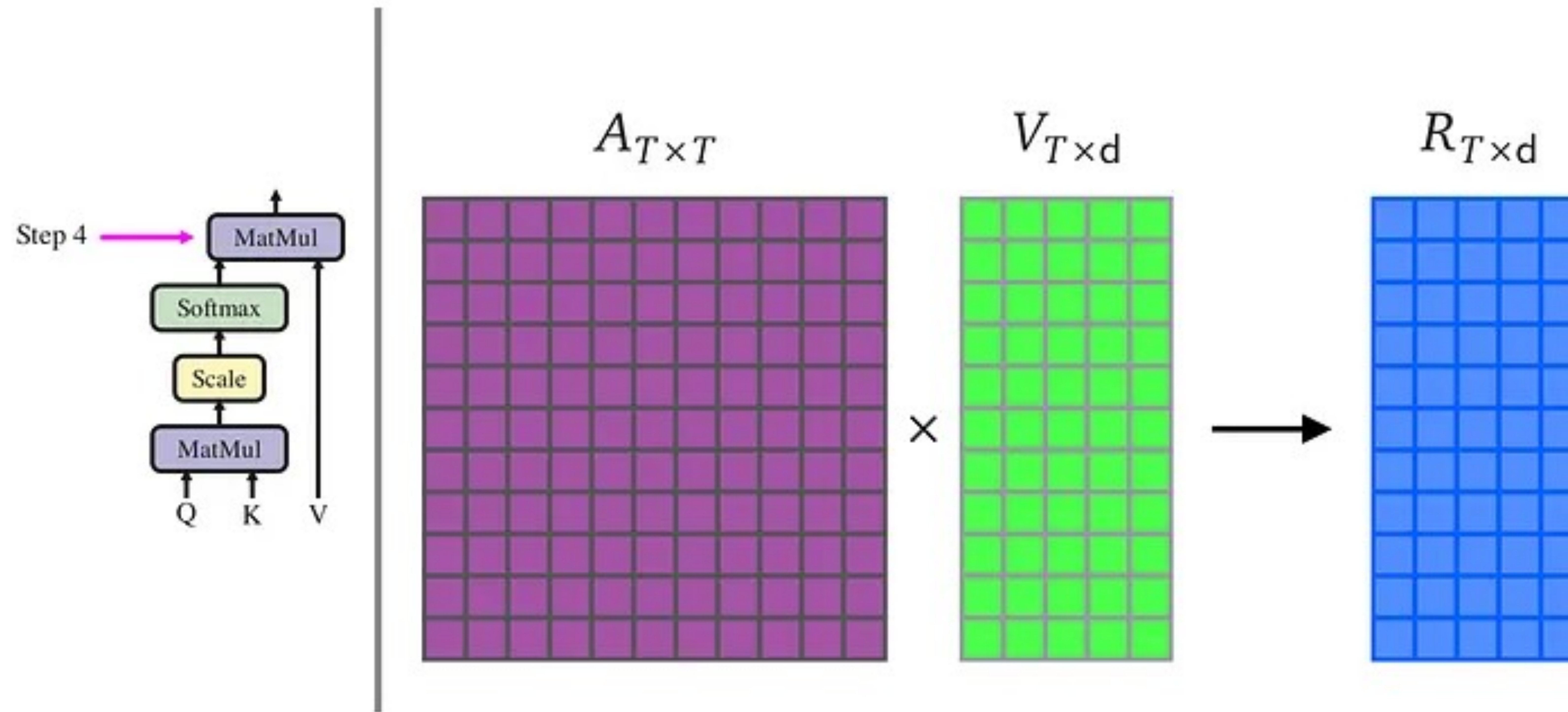
	a → $\vec{E}_1$ → $\vec{Q}_1$	fluffy → $\vec{E}_2$ → $\vec{Q}_2$	blue → $\vec{E}_3$ → $\vec{Q}_3$	creature → $\vec{E}_4$ → $\vec{Q}_4$	roamed → $\vec{E}_5$ → $\vec{Q}_5$	the → $\vec{E}_6$ → $\vec{Q}_6$	verdant → $\vec{E}_7$ → $\vec{Q}_7$	forest → $\vec{E}_8$ → $\vec{Q}_8$
a → $\vec{E}_1 \xrightarrow{W_k} \vec{K}_1$	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fluffy → $\vec{E}_2 \xrightarrow{W_k} \vec{K}_2$	0.00	1.00	0.00	0.42	0.00	0.00	0.00	0.00
blue → $\vec{E}_3 \xrightarrow{W_k} \vec{K}_3$	0.00	0.00	1.00	0.58	0.00	0.00	0.00	0.00
creature → $\vec{E}_4 \xrightarrow{W_k} \vec{K}_4$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
roamed → $\vec{E}_5 \xrightarrow{W_k} \vec{K}_5$	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
the → $\vec{E}_6 \xrightarrow{W_k} \vec{K}_6$	0.00	0.00	0.00	0.00	0.99	1.00	0.00	0.00
verdant → $\vec{E}_7 \xrightarrow{W_k} \vec{K}_7$	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00
forest → $\vec{E}_8 \xrightarrow{W_k} \vec{K}_8$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

(Each column: $\vec{E}_i \xrightarrow{W_Q} \vec{Q}_i$)

Step 2 - Scale by $\sqrt{d_k}$

Scaled Dot Product Attention

The Process



Step 4 - Weighted sum of values

Each output vector is a convex combination of the value vectors, weighted by how much each query attended to each key.

Scaled Dot Product Attention

An Example

- Consider the sentence: “The cat sat” with 3 tokens.
- After embedding, X is a $3 \times d$ matrix. We compute Q, K, V via learned projections.
- The **score matrix** $S = QK^T$ might look like

	The	cat	sat
The	[1.2	0.3	-0.5]
cat	[0.8	2.1	0.4]
sat	[-0.2	1.5	0.9]

Scaled Dot Product Attention

An Example

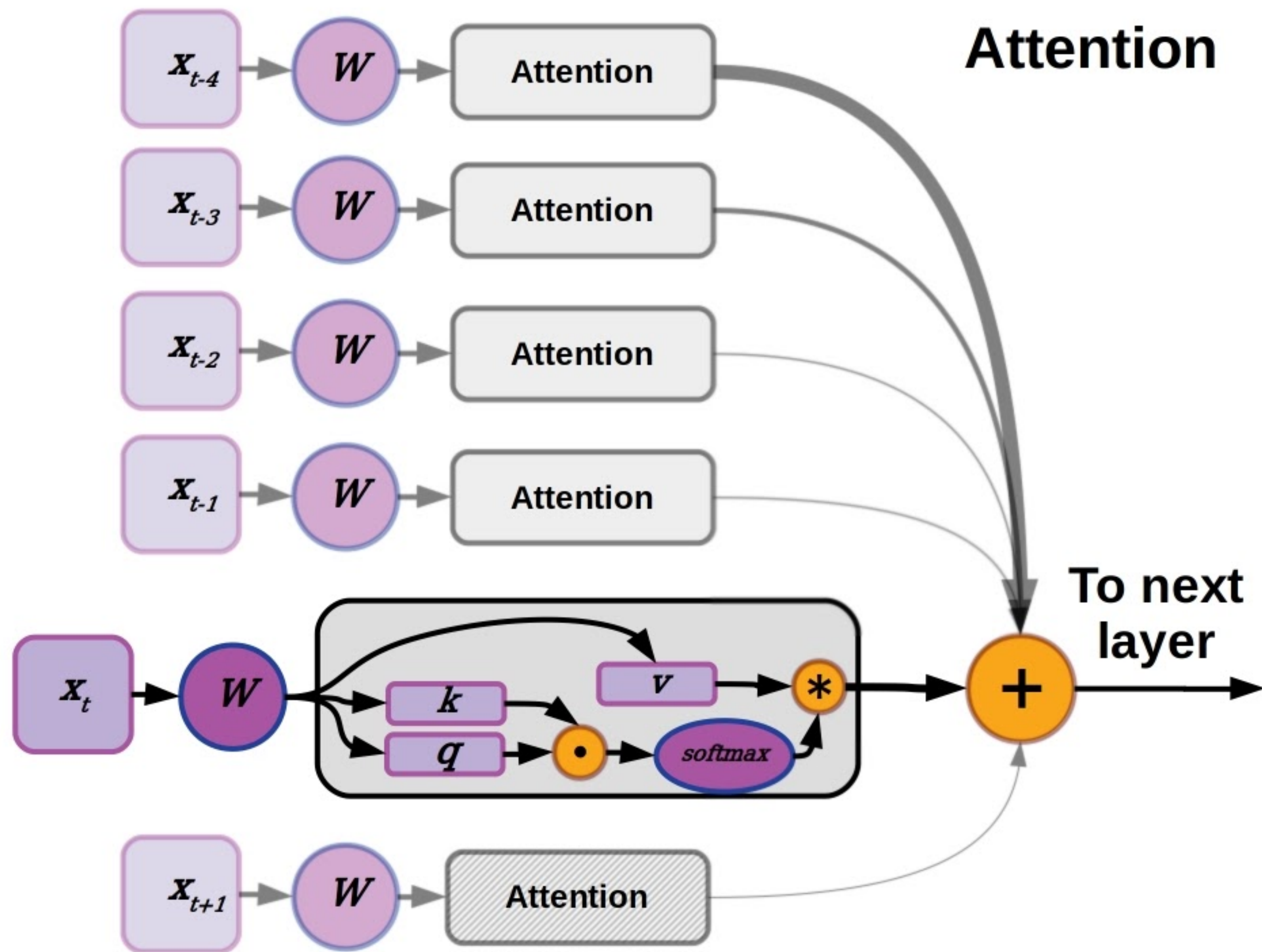
- Consider the sentence: “The cat sat” with 3 tokens.
- After embedding, X is a $3 \times d$ matrix. We compute Q, K, V via learned projections.
- **After scaling and softmax** (row-wise), attention weights A:

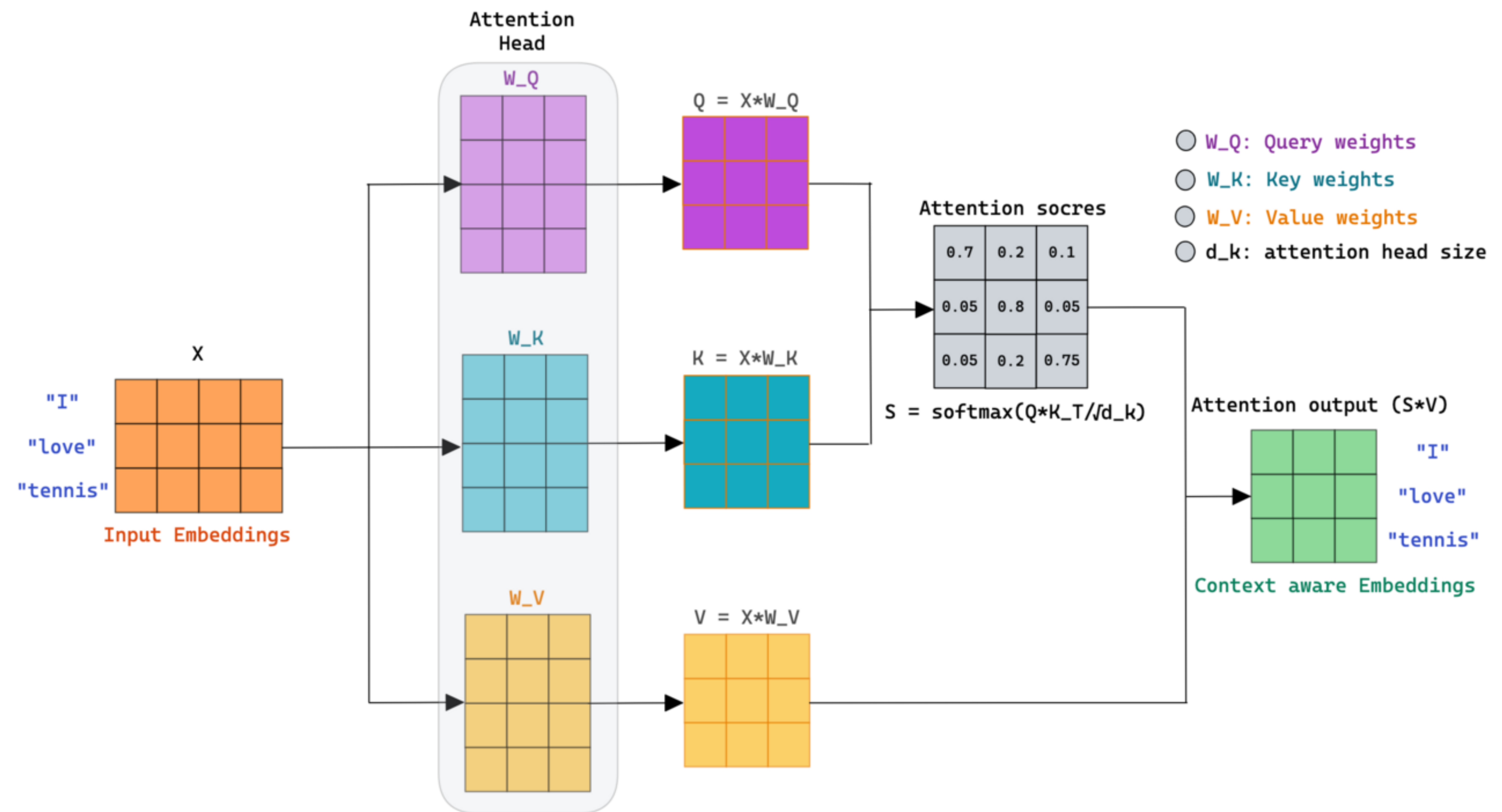
	The	cat	sat	
The	[0.70	0.23	0.07]	← "The" mostly attends to itself
cat	[0.18	0.69	0.13]	← "cat" strongly attends to itself
sat	[0.08	0.57	0.35]	← "sat" mostly attends to "cat" (subject-verb!)

Positional Encoding

- Attention is **permutation invariant**
  - If you shuffle the tokens, the same pairs attend to each other, just in different order. **The model has no idea which token comes first**.
- This is a fundamental problem: “The cat chased the dog” and “The dog chased the cat” have the same tokens, just reordered.
- We need to inject positional information into the token representations.
- $\hat{X} = X + PE$ (element-wise addition)

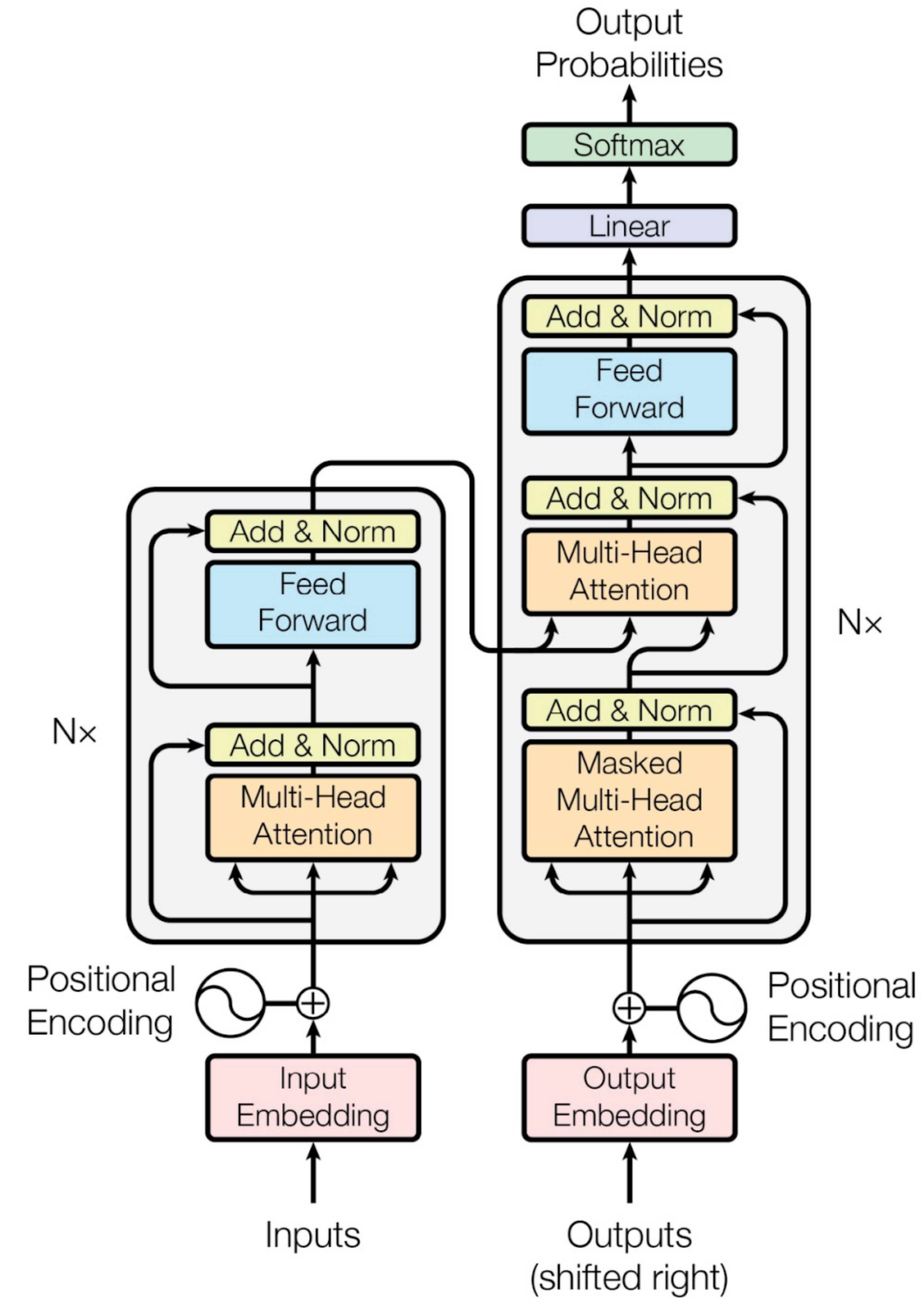
Transformers





Transformers

x → [Self-Attention] → [Add & Norm] → [Feed-Forward] → [Add & Norm] → output



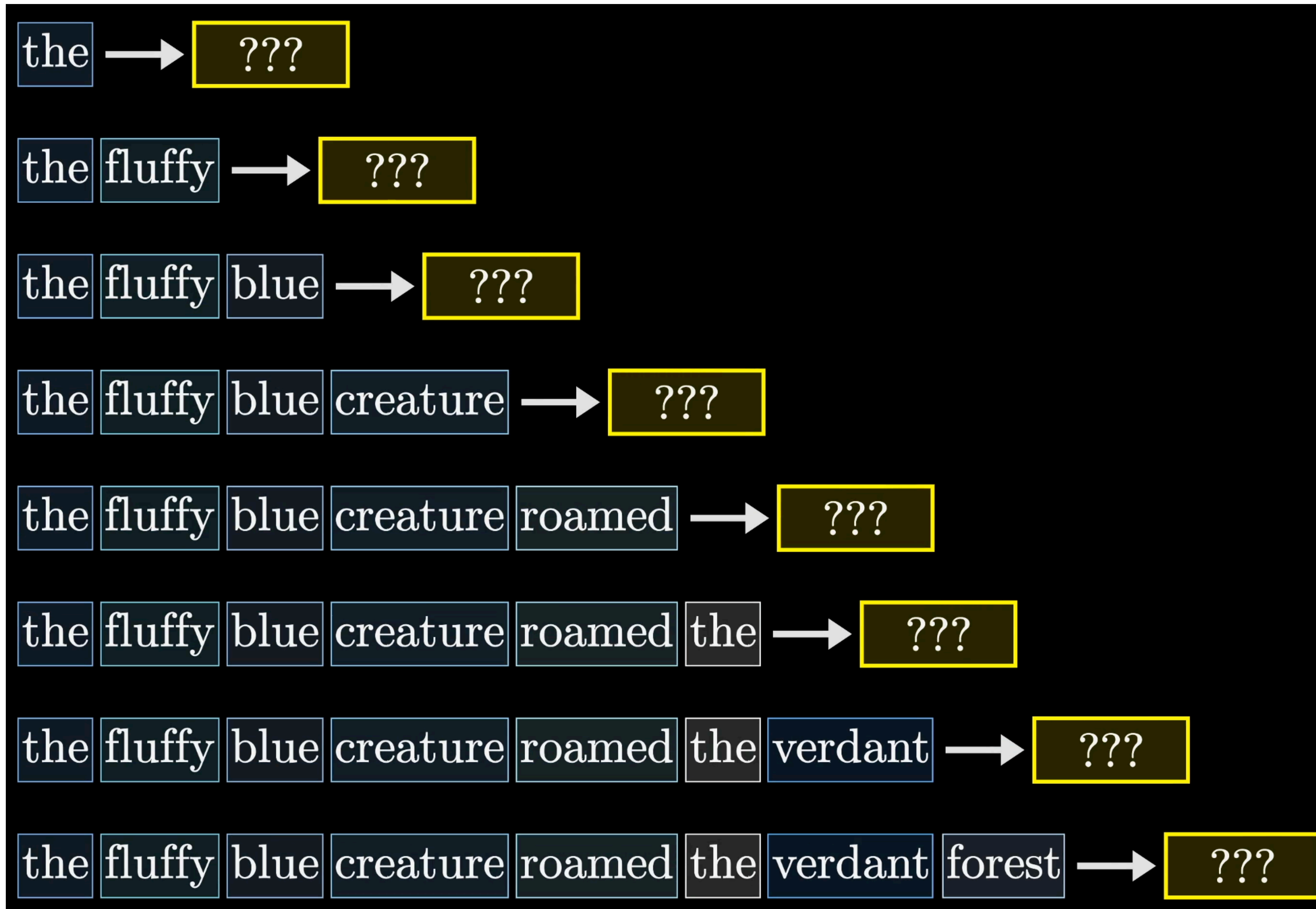
Transformers

- The Language Modeling Objective
  - Train the model to predict the next token given all previous tokens
  - This is a sort of multi-class classification task
  - Use cross entropy loss

$$L = -\sum_t logP(x_t | x_1, x_2, \ldots, x_{t-1})$$

Transformers

Teacher Forcing



Transformers

Causal (Masked) Self-Attention

- The critical modification for a **language model**:
  - A token **cannot** attend to **future** tokens - that would be cheating (you'd be predicting a token using the token itself as input).
  - Enforce this with a causal mask - set all attention scores above the diagonal to $-\infty$ before softmax:

$$
\begin{array}{c|cccc}
 & x_1 & x_2 & x_3 & x_4 \\
x_1 & [\; s_{11} & -\infty & -\infty & -\infty \;] \\
x_2 & [\; s_{21} & s_{22} & -\infty & -\infty \;] \\
x_3 & [\; s_{31} & s_{32} & s_{33} & -\infty \;] \\
x_4 & [\; s_{41} & s_{42} & s_{43} & s_{44} \;]
\end{array}
$$

  - After softmax, $-\infty \rightarrow 0$, so future positions contribute nothing to the weighted sum. Each token attends to itself and all previous tokens only.

Transformers

Teacher Forcing

- During training, we always feed the model the ground truth tokens as input, **regardless** of what the model predicted at the previous step. This is called teacher forcing:

  Input: [<BOS>, "The", "cat", "sat"]

  Target: ["The", "cat", "sat", "on"]

  ↑ shift by one position

- The loss is cross-entropy between predicted logits and target token IDs, averaged over all positions:

$$L = -\sum_{t} logP(x_t | x_1, x_2, \ldots, x_{t-1})$$

Transformers

Causal (Masked) Self-Attention

- Key efficiency insight:
  - During training, you can compute all **T** next-token predictions in a single forward pass using this mask
  - You don't need to run T separate forward passes.
  - This is why transformers are so much faster to train than RNNs (which require T **sequential** steps).

Transformers

Causal (Masked) Self-Attention

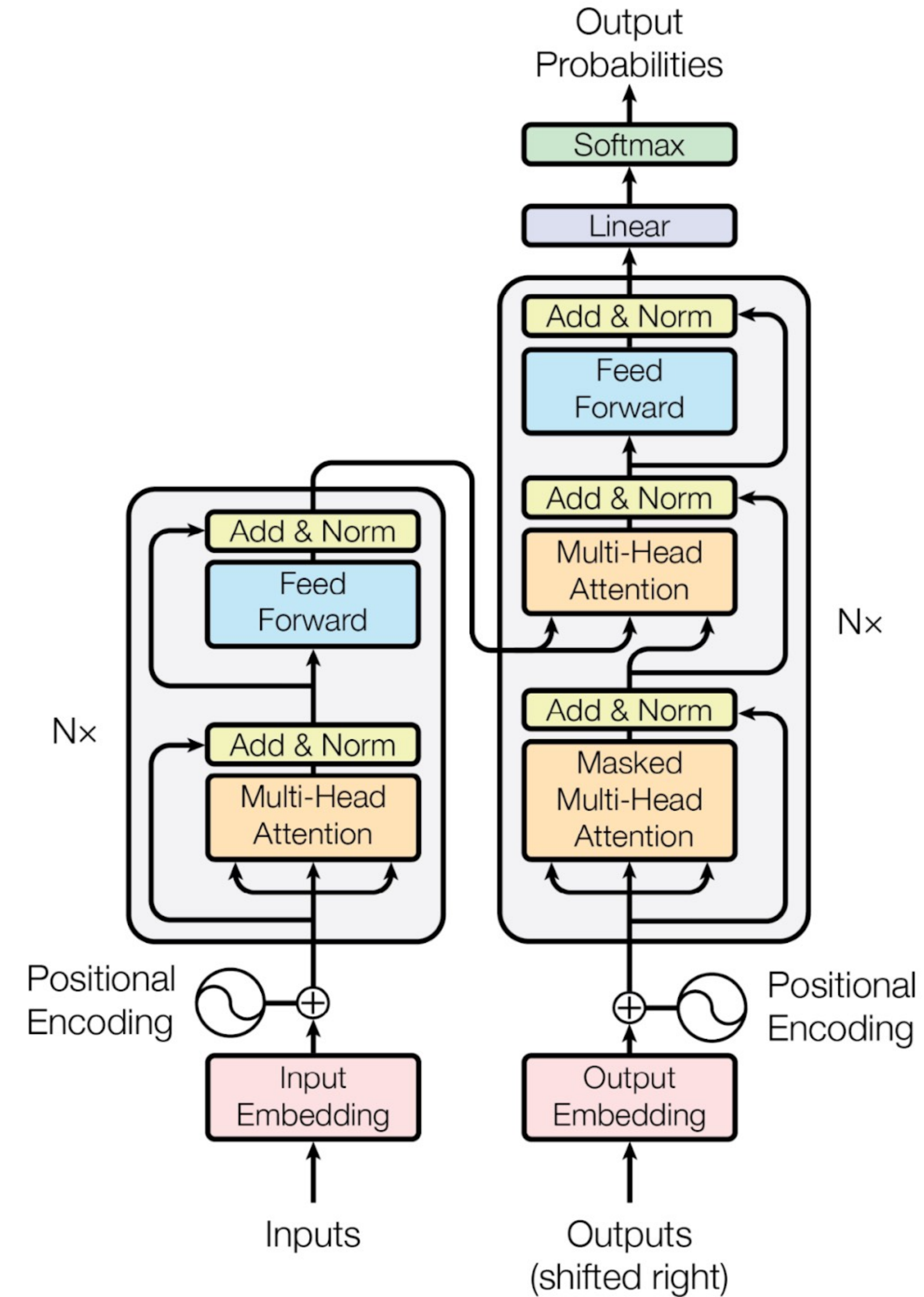
```
# Pseudocode for one transformer block (pre-norm style)

def transformer_block(x):
    # --- Self-Attention Sublayer ---
    x_norm = layer_norm(x)                      # normalize first
    Q = x_norm @ W_Q                            # (n × d) @ (d × dₖ) = (n × dₖ)
    K = x_norm @ W_K
    V = x_norm @ W_V
    scores = (Q @ K.T) / sqrt(d_k)             # (n × n) raw attention scores
    scores = apply_causal_mask(scores)          # set upper triangle to -inf
    weights = softmax(scores, dim=-1)           # (n × n) attention weights
    attn_out = weights @ V                      # (n × dᵥ) weighted values
    attn_out = attn_out @ W_O                  # (n × d) project back
    x = x + attn_out                           # residual connection

    # --- Feed-Forward Sublayer ---
    x_norm = layer_norm(x)
    ffn_out = relu(x_norm @ W_1 + b_1) @ W_2 + b_2   # expand then contract
    x = x + ffn_out                            # residual connection

    return x
```

GPT-2 Small: Concrete Numbers

Hyperparameter	Value
Layers (L)	12
Model Dimension (d)	768
Attention Heads (12)	12
Head Dimension	64
FFN Dimension	3072
Vocab Size	50,247
Max Context Length	1024
Total Params	117M Params

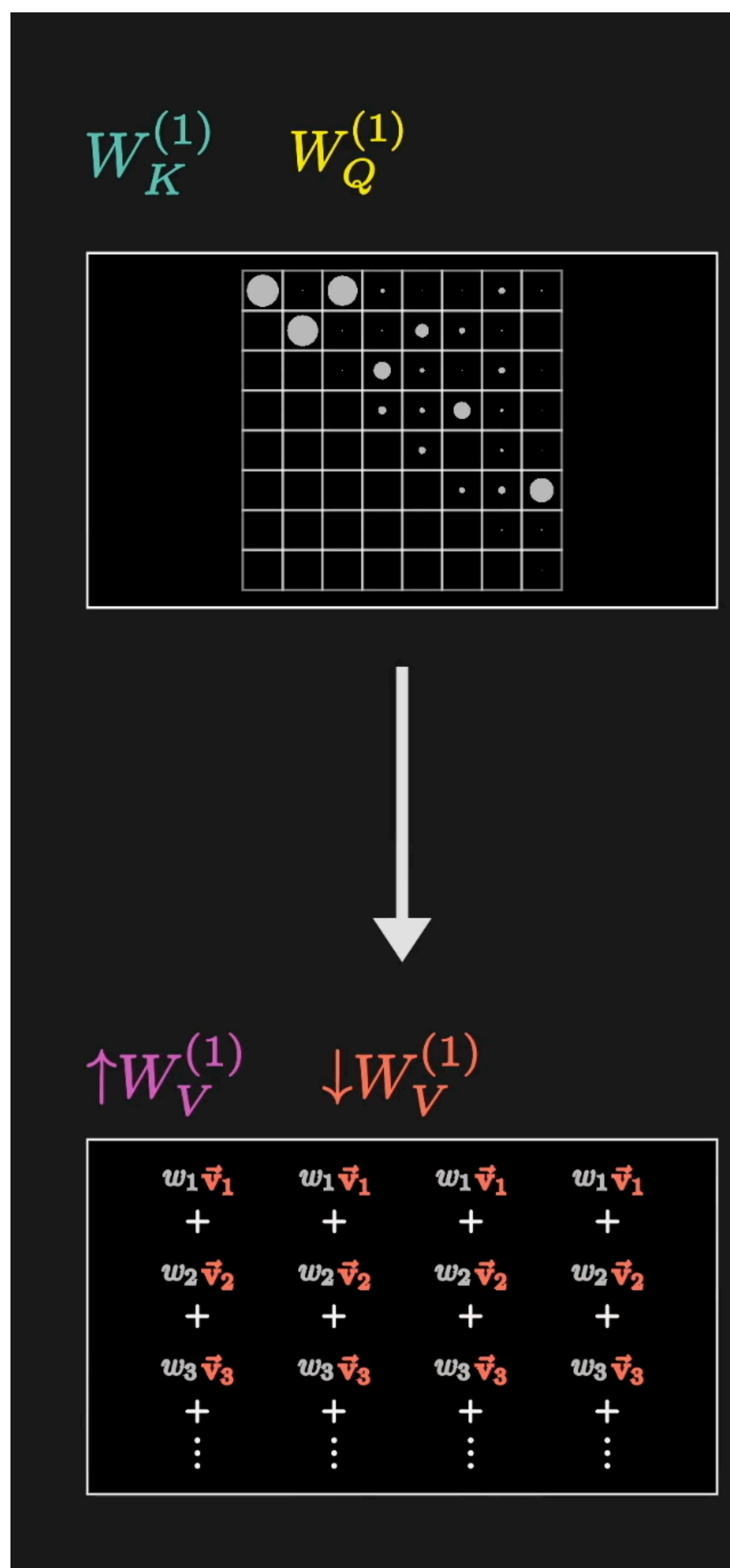
Transformers

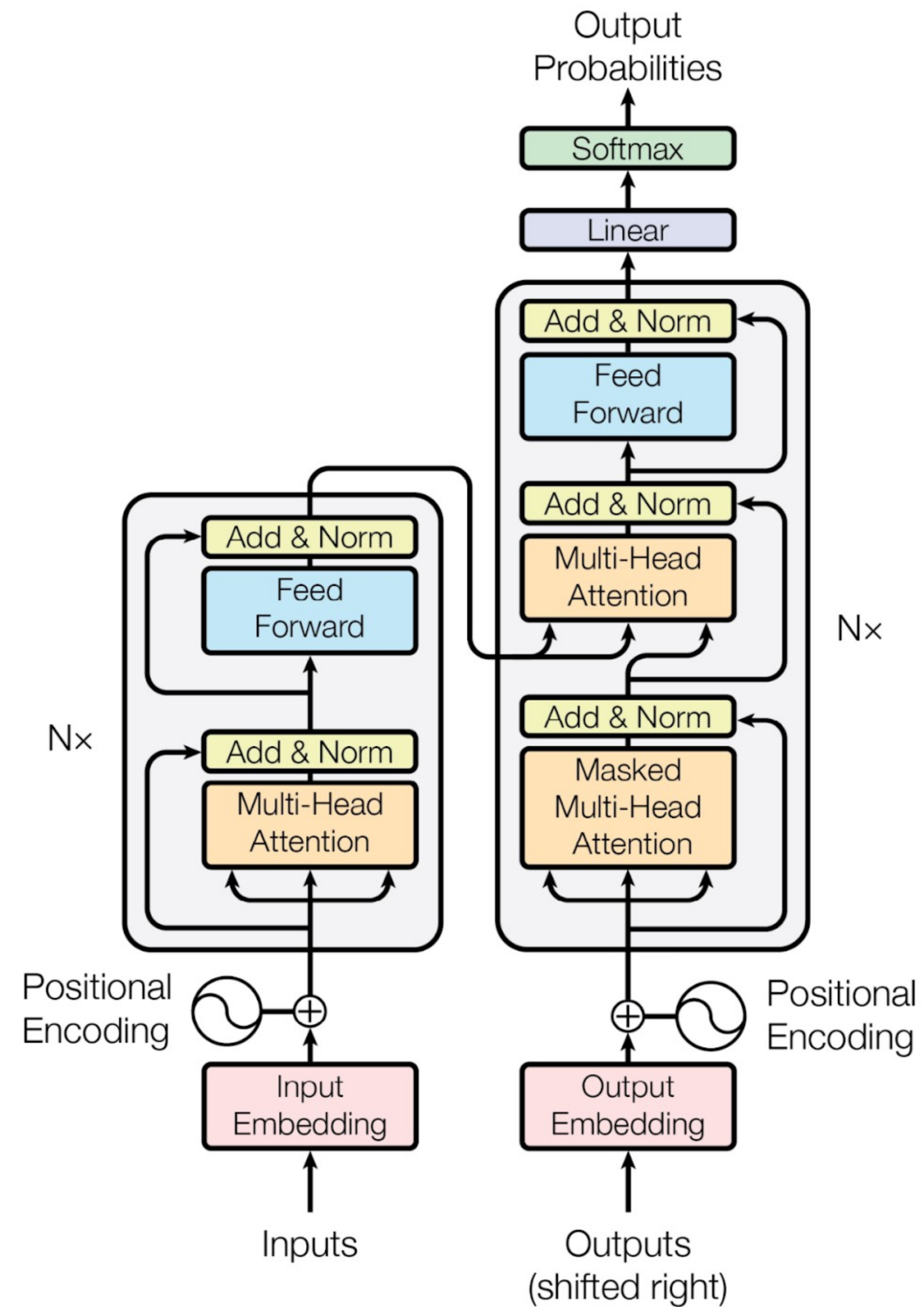
Encoder-Decoder vs. Decoder Only



Transformers

Multi Head Attention





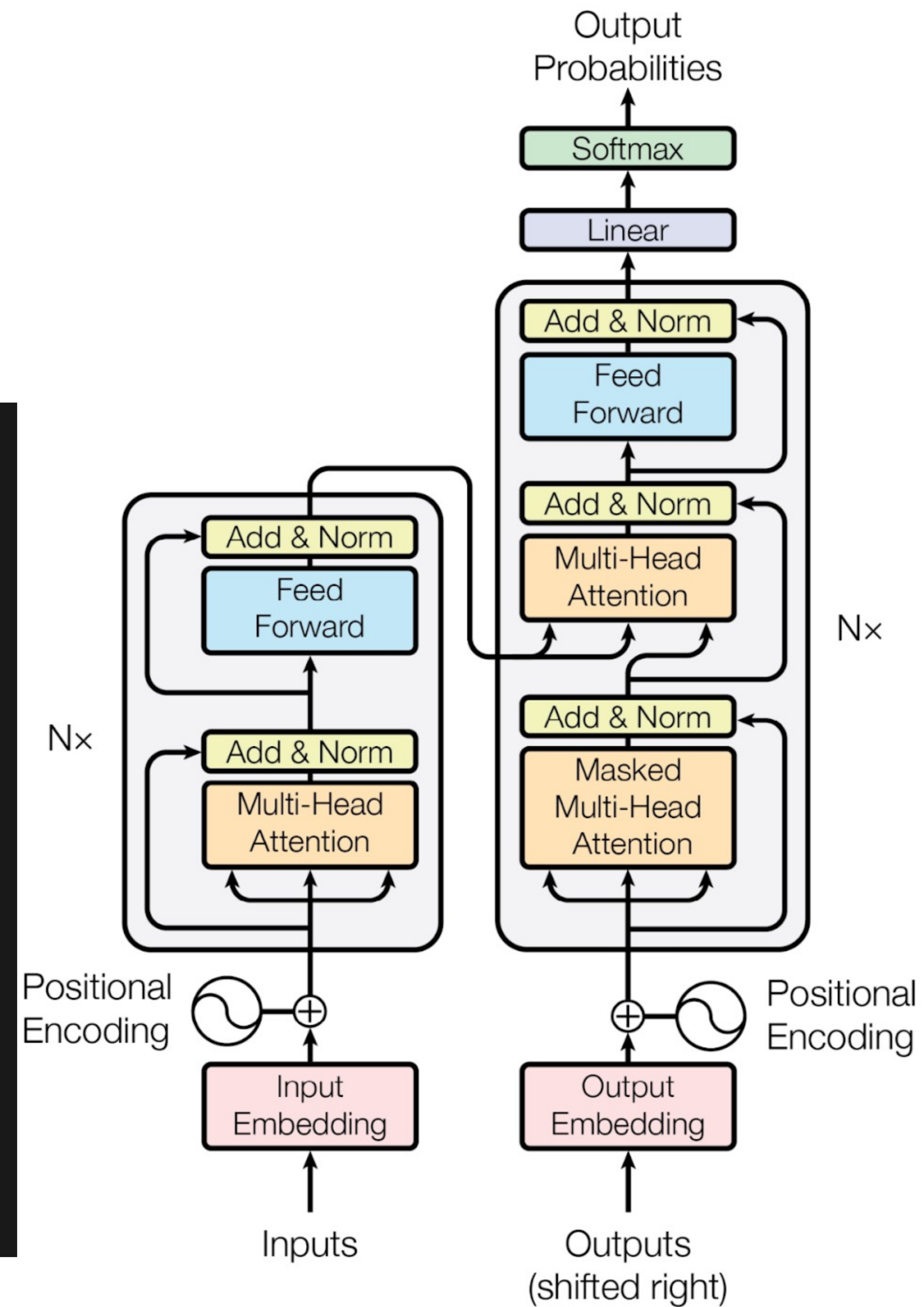
Transformers

Multi Head Attention

$W_K^{(1)}$ $W_Q^{(1)}$ $W_K^{(2)}$ $W_Q^{(2)}$ $W_K^{(3)}$ $W_Q^{(3)}$ $W_K^{(4)}$ $W_Q^{(4)}$ . . .

$\uparrow W_V^{(1)}$ $\downarrow W_V^{(1)}$ $\uparrow W_V^{(2)}$ $\downarrow W_V^{(2)}$ $\uparrow W_V^{(3)}$ $\downarrow W_V^{(3)}$ $\uparrow W_V^{(4)}$ $\downarrow W_V^{(4)}$ . . .

$w_1\vec{v}_1 + w_2\vec{v}_2 + w_3\vec{v}_3 + \cdots$



Recommender Systems

- **The Goal:** To predict the “utility” (rating, click, purchase) a user u would assign to an item i.

Recommender Systems

- **The Goal:** To predict the "utility" (rating, click, purchase) a user u would assign to an item i.
- **The Data Structure:**
  - We represent the system as a User-Item Rating Matrix R of size $m \times n$, where:
  - $m =$ number of users
  - $n =$ number of items
  - r_{ui} is the rating given by user u to item i

Recommender Systems

- **The Goal:** To predict the "utility" (rating, click, purchase) a user u would assign to an item i.
- **The Data Structure:**
  - We represent the system as a User-Item Rating Matrix R of size $m \times n$, where:
  - $m =$ number of users
  - $n =$ number of items
  - r_{ui} is the rating given by user u to item i
- **The Challenge:** In real-world scenarios, R is **extremely sparse**.
  - Most users only interact with a tiny fraction of available items

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating   - rating between 1 to 5

Recommender Systems

Taxonomy of Recommender Systems

- **Content-Based Filtering (CBF)**
  - **Concept**: “Show me more of what I liked before”
  - **Mechanism**: Focuses on the properties (features) of the items.
  - **Item Profile**: An item i is represented by a feature vector $\vec{x}_i$
    - (e.g., for a movie: $[$action, comedy, duration, director_id$]$).
  - **User Profile**: A user u is represented by a vector $\vec{x}_u$
    - often calculated as the weighted average of the profiles of items they liked.

Recommender Systems

Taxonomy of Recommender Systems

- **Content-Based Filtering (CBF)**
  - **Concept**: “Show me more of what I liked before”
  - **Mechanism**: Focuses on the properties (features) of the items.
  - **Item Profile**: An item i is represented by a feature vector $\vec{x}_i$
    - (e.g., for a movie: [action, comedy, duration, director_id]).
  - **User Profile**: A user u is represented by a vector $\vec{x}_u$
    - often calculated as the weighted average of the profiles of items they liked.
- **Mathematical Core: Similarity Measures**
  - To recommend, we calculate the **similarity** between the User Profile and Item Profiles using Cosine Similarity:

$$textsim(\vec{x}_u, \vec{x}_i) = \frac{\vec{x}_u \cdot \vec{x}_i}{||\vec{x}_u|| \, ||\vec{x}_i||}$$

    - Pros: No “Cold Start” problem for new items; highly interpretable.
    - Cons: “Filter Bubbles” (no serendipity/discovery); limited by the quality of item metadata.

Recommender Systems

Taxonomy of Recommender Systems

- **Collaborative Filtering (CF)**
  - **Concept**: "People who liked this also liked that"
    - CF **ignores** item features and relies solely on **user-item interactions**

Recommender Systems

Taxonomy of Recommender Systems

- **Collaborative Filtering (CF)**
  - **Concept**: “People who liked this also liked that”
    - CF **ignores** item features and relies solely on **user-item interactions**
  - **Memory-Based CF (Neighborhood Methods)**
    - This uses the actual data in R to find “neighbors”
    - **User-User CF**: Find users v similar to u.

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u)} \text{sim}(u, v) \cdot (r_{vi} - \bar{r}_v)}{\sum_{v \in N(u)} |\text{sim}(u, v)|}$$

Where $\bar{r}_v$ is the average rating of user v to account for “optimistic” vs “pessimistic” raters

Recommender Systems

Taxonomy of Recommender Systems

- **Collaborative Filtering (CF)**
  - **Concept**: “People who liked this also liked that”
    - CF **ignores** item features and relies solely on **user-item interactions**
  - **Memory-Based CF (Neighborhood Methods)**
    - This uses the actual data in R to find “neighbors”
    - **User-User CF**: Find users v similar to u.

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u)} \text{sim}(u, v) \cdot (r_{vi} - \bar{r}_v)}{\sum_{v \in N(u)} |\text{sim}(u, v)|}$$

Where $\bar{r}_v$ is the average rating of user v to account for “optimistic” vs “pessimistic” raters

- **Item-Item CF**: Find items j similar to i based on how users rated them.

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum_{j \in N(i)} \text{sim}(i, j)}$$

Item-Item is generally **more stable in production**
item characteristics change slower than user tastes

users (columns) / movies (rows)

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		2.6	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Predict by taking weighted average:

$r_{1.5}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

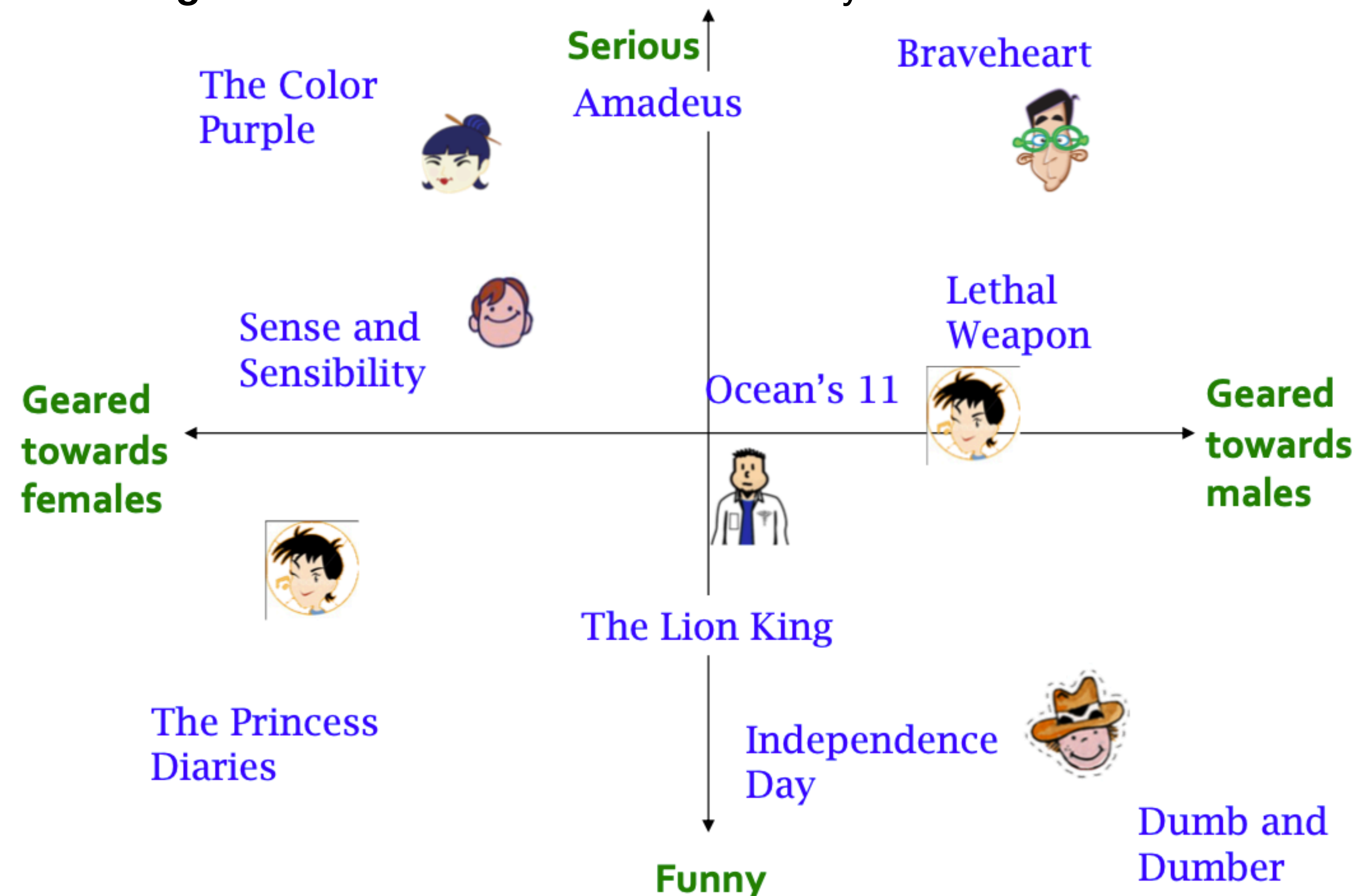
If you are doing **user-user CF**, then you compute the mean for the user and subtract it from the ratings of the items that they have rated.

If you are doing **item-item CF**, then you compute the mean for the item and subtract it from the ratings of the users who rated that item.

Recommender Systems

Taxonomy of Recommender Systems

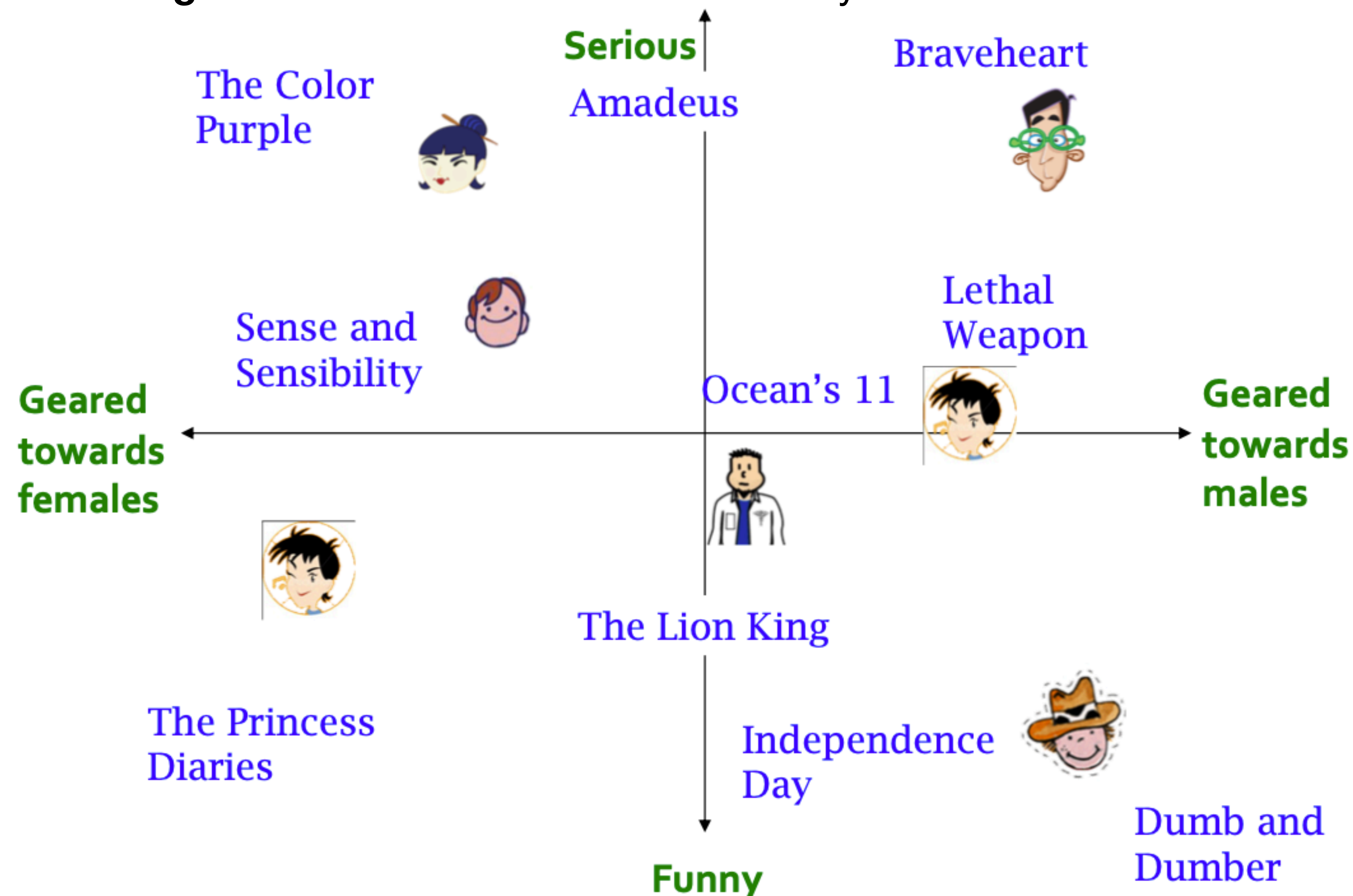
- **Collaborative Filtering (CF) - Model Based**
  - **Concept**: “People who liked this also liked that”
    - CF **ignores** item features and relies solely on **user-item interactions**

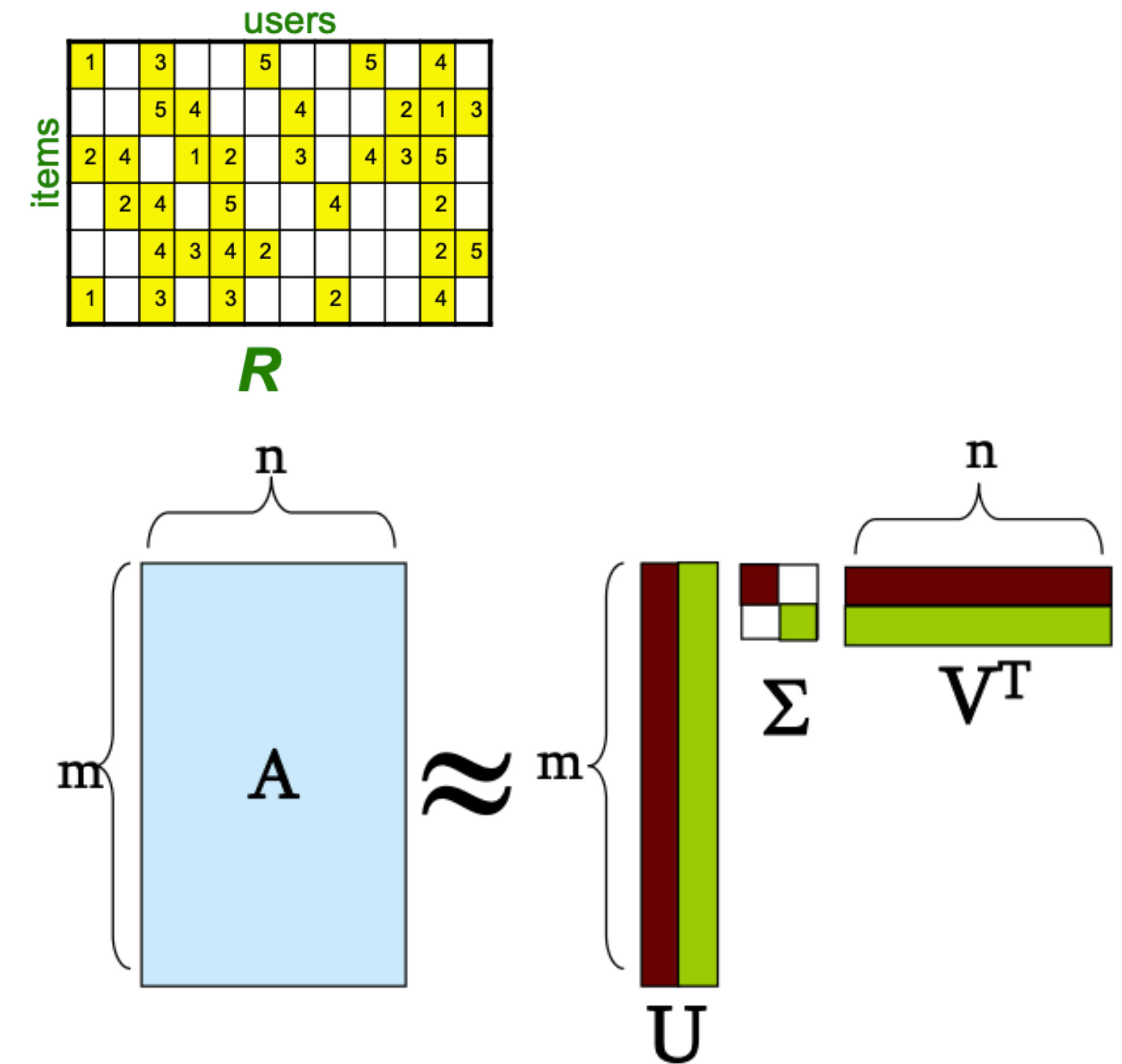


Recommender Systems

Taxonomy of Recommender Systems

- **Collaborative Filtering (CF) - Model Based**
  - **Concept**: “People who liked this also liked that”
    - CF **ignores** item features and relies solely on **user-item interactions**

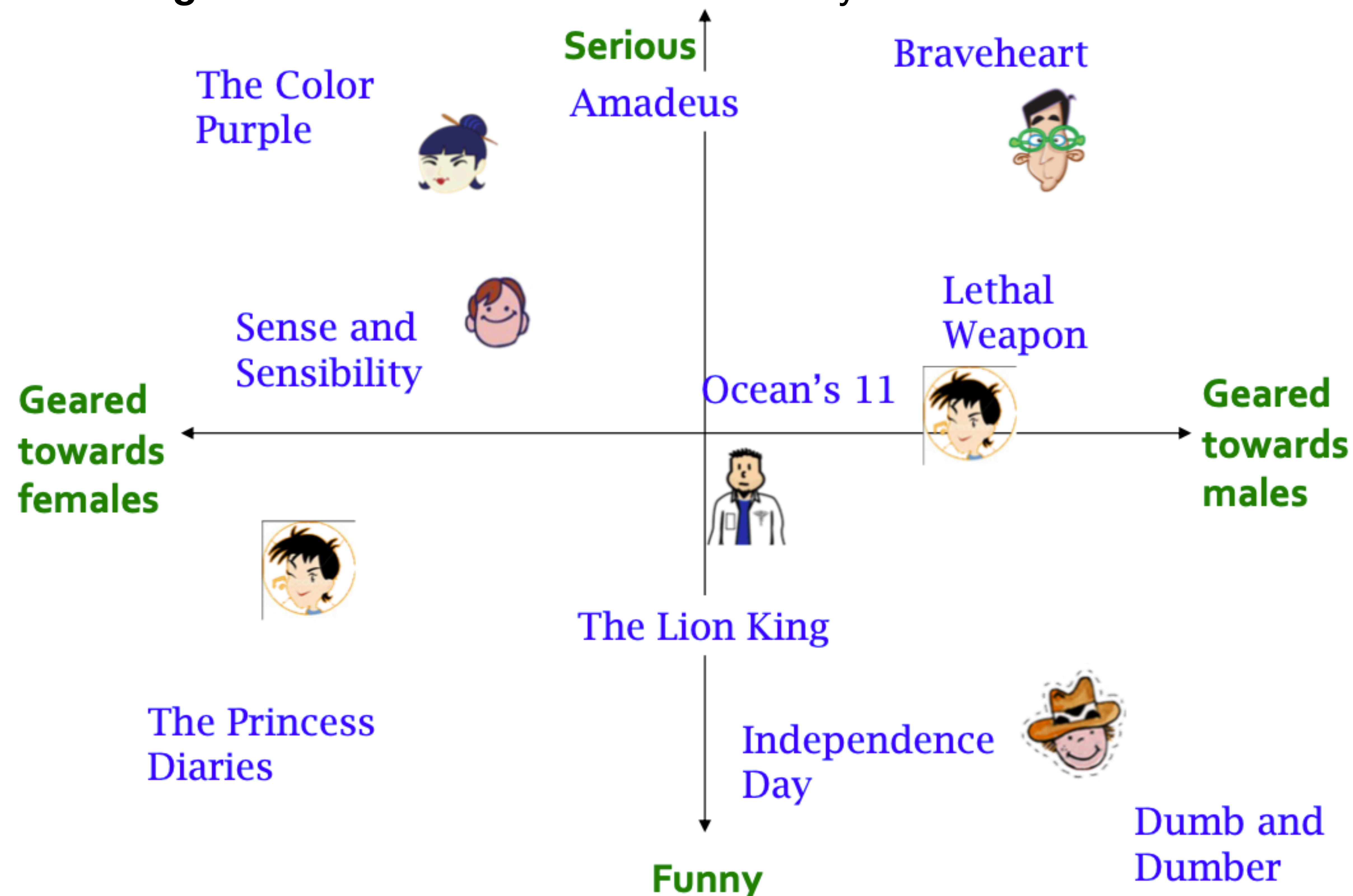




Recommender Systems

Taxonomy of Recommender Systems

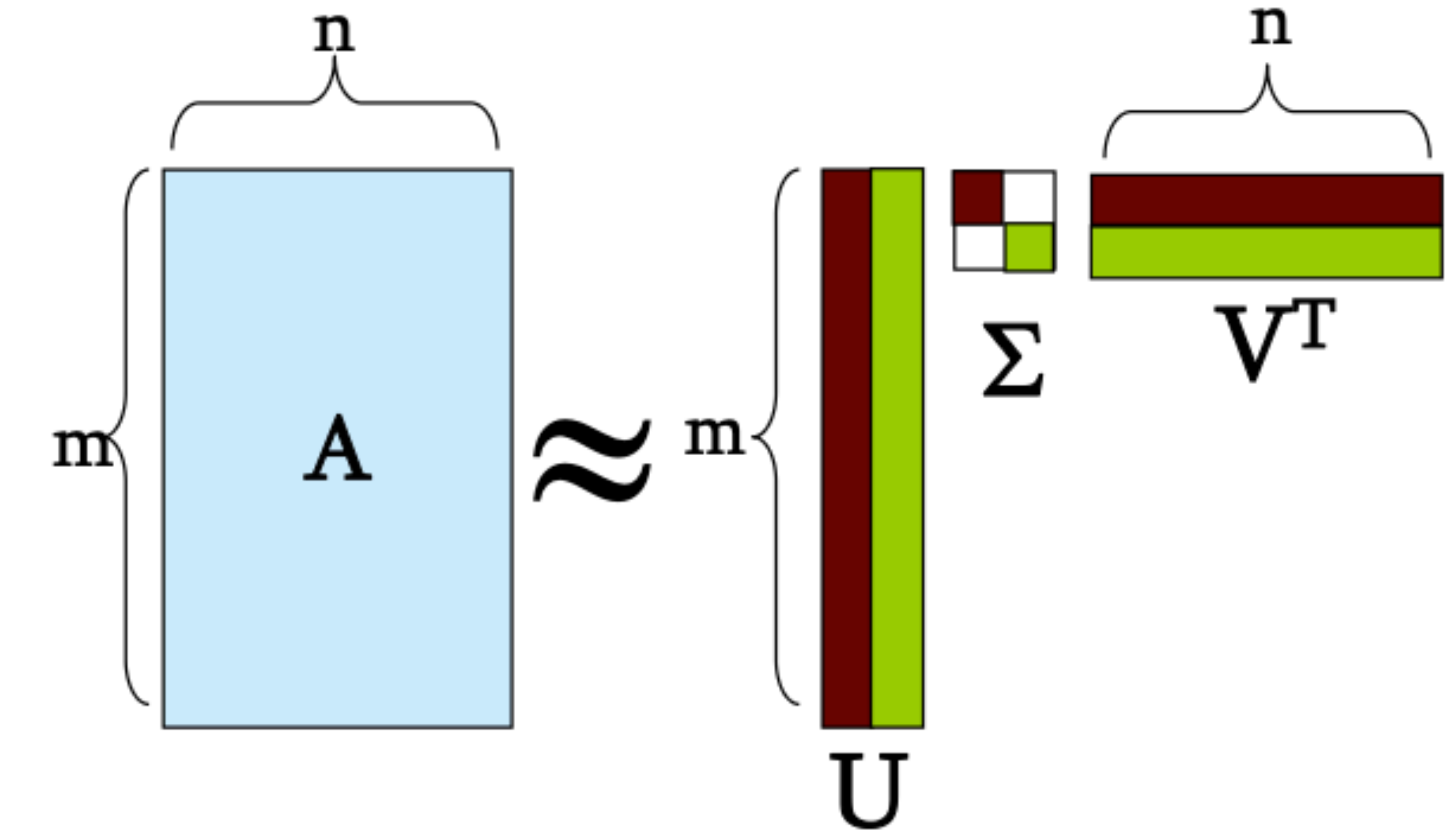
- **Collaborative Filtering (CF) - Model Based**
  - **Concept**: “People who liked this also liked that”
    - CF **ignores** item features and relies solely on **user-item interactions**



users (columns) × items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R



users (columns) × items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R ≈

factors (columns) × items (rows)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

users (columns) × factors (rows)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

P^T

Recommender Systems

Taxonomy of Recommender Systems

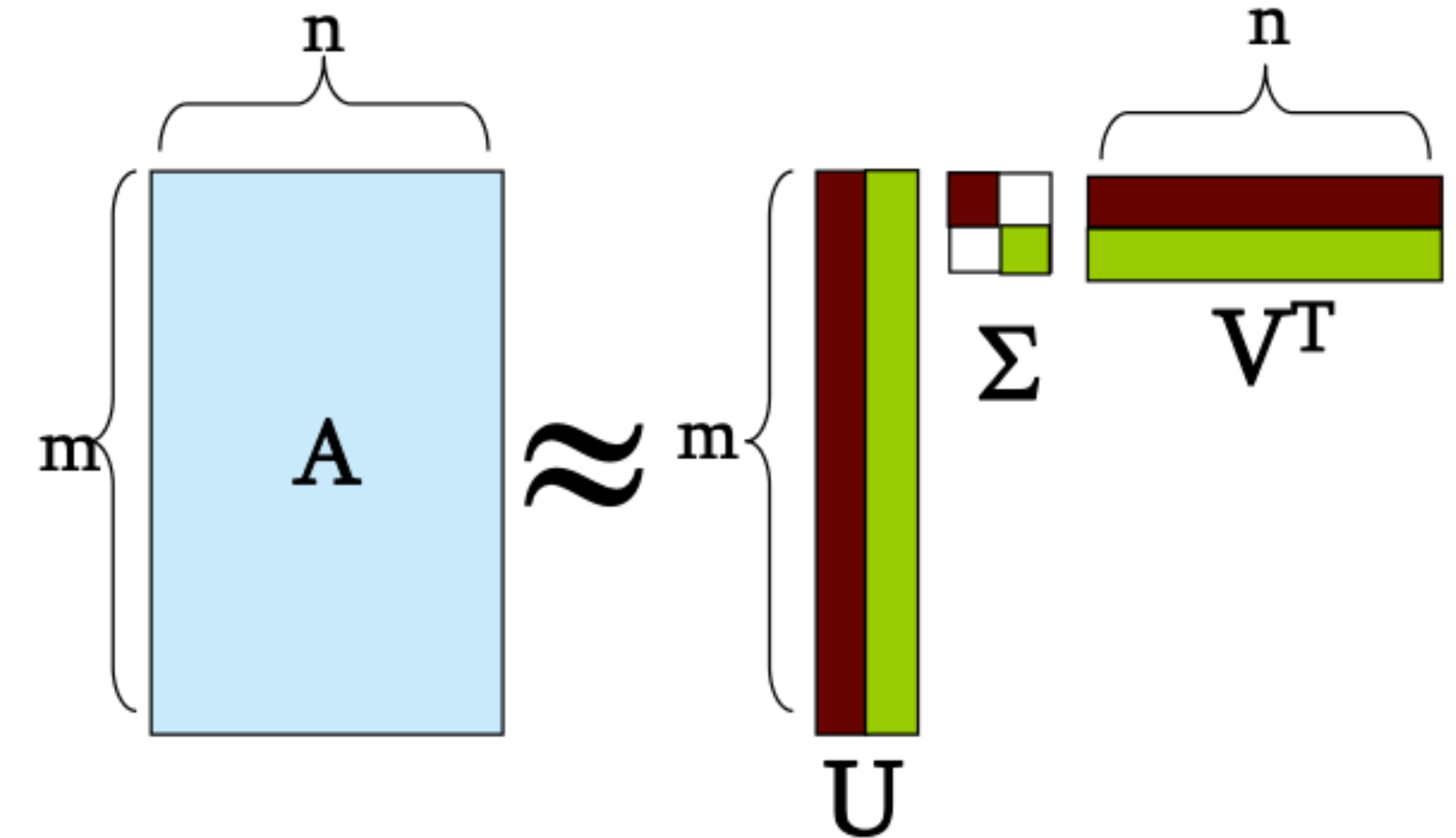
- **Model Based (Matrix Factorization)**
  - We assume there are k hidden (latent) factors that explain the ratings.
  - **The Math:**
    - We decompose the sparse matrix R into two lower-rank matrices:
      - $P \in \mathbb{R}^{m \times k}$ (User-Latent Factor matrix)
      - $Q \in \mathbb{R}^{n \times k}$ (Item-Latent Factor matrix)
    - The predicted rating is the dot product:

$$\hat{r}_{ui} = q_i^T p_u = \sum_{f=1}^{k} p_{uf} \cdot q_{if}$$

users (columns) / items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R



users (columns) / items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R ≈

factors (columns) / items (rows)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

users (columns) / factors (rows)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

P^T

Recommender Systems

Taxonomy of Recommender Systems

- **Model Based (Matrix Factorization)**
  - We assume there are k hidden (latent) factors that explain the ratings.
  - **The Math:**
    - We decompose the sparse matrix R into two lower-rank matrices:
      - $P \in \mathbb{R}^{m \times k}$ (User-Latent Factor matrix)
      - $Q \in \mathbb{R}^{n \times k}$ (Item-Latent Factor matrix)
    - The predicted rating is the dot product:

$$\hat{r}_{ui} = q_i^T p_u = \sum_{f=1}^{k} p_{uf} \cdot q_{if}$$

  - **The Optimization Problem** - How do we learn P and Q?
    - We want to find P and Q that minimize the difference between the actual ratings and our predictions.
    - We use **Regularized** Squared Error to prevent overfitting

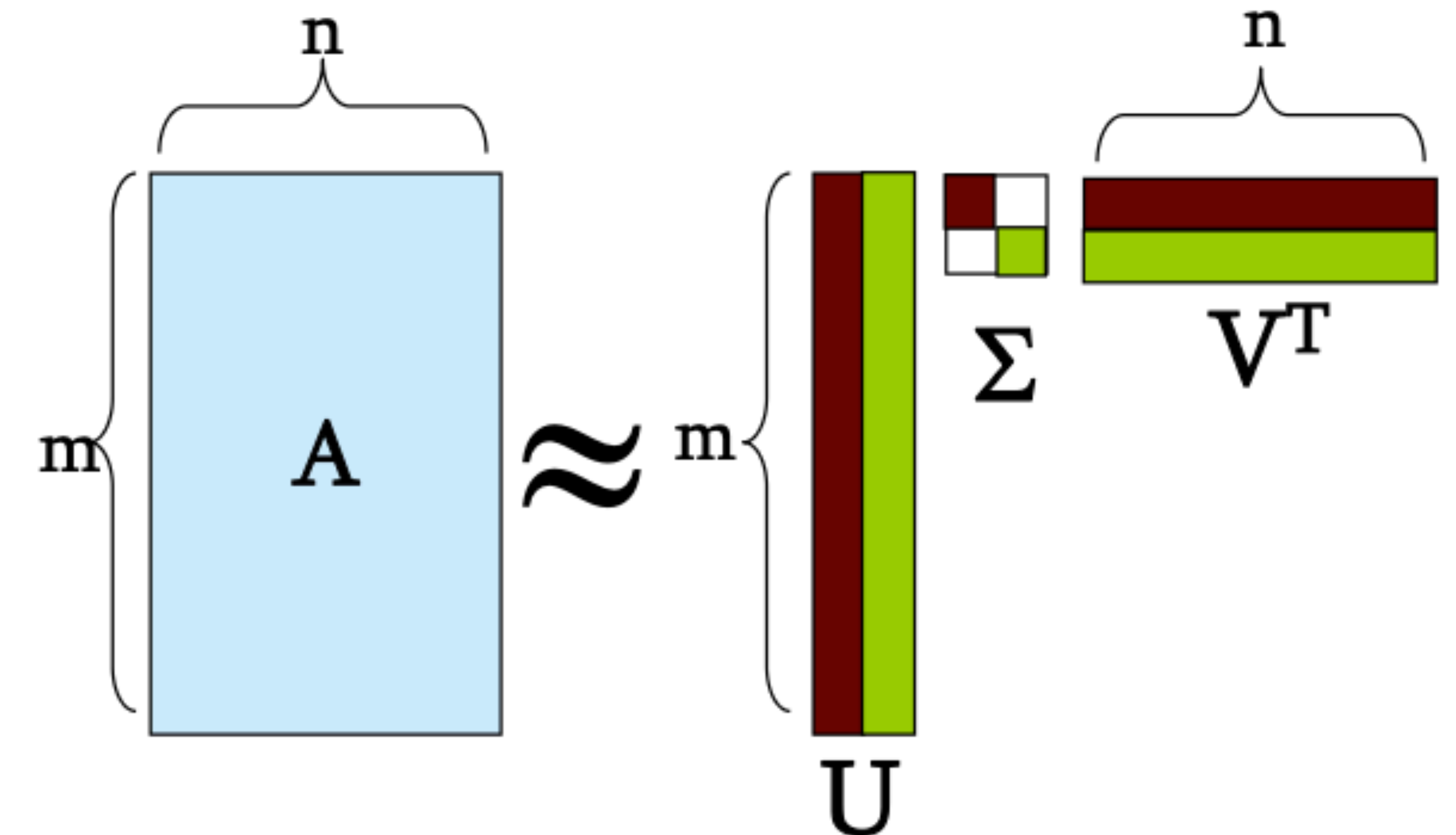
$$\ell = min_{P,Q} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

    - Where:
      - $\mathcal{K}$ is the set of (user, item) pairs **for which we have ratings.**

users (columns), items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R



users (columns), items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R ≈

factors (columns), items (rows)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

users (columns), factors (rows)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

P^T

Recommender Systems

Taxonomy of Recommender Systems

- **Model Based (Matrix Factorization)**
    - **The Optimization Problem** - How do we learn P and Q?
        - We want to find P and Q that minimize the difference between the actual ratings and our predictions.
        - We use **Regularized** Squared Error to prevent overfitting

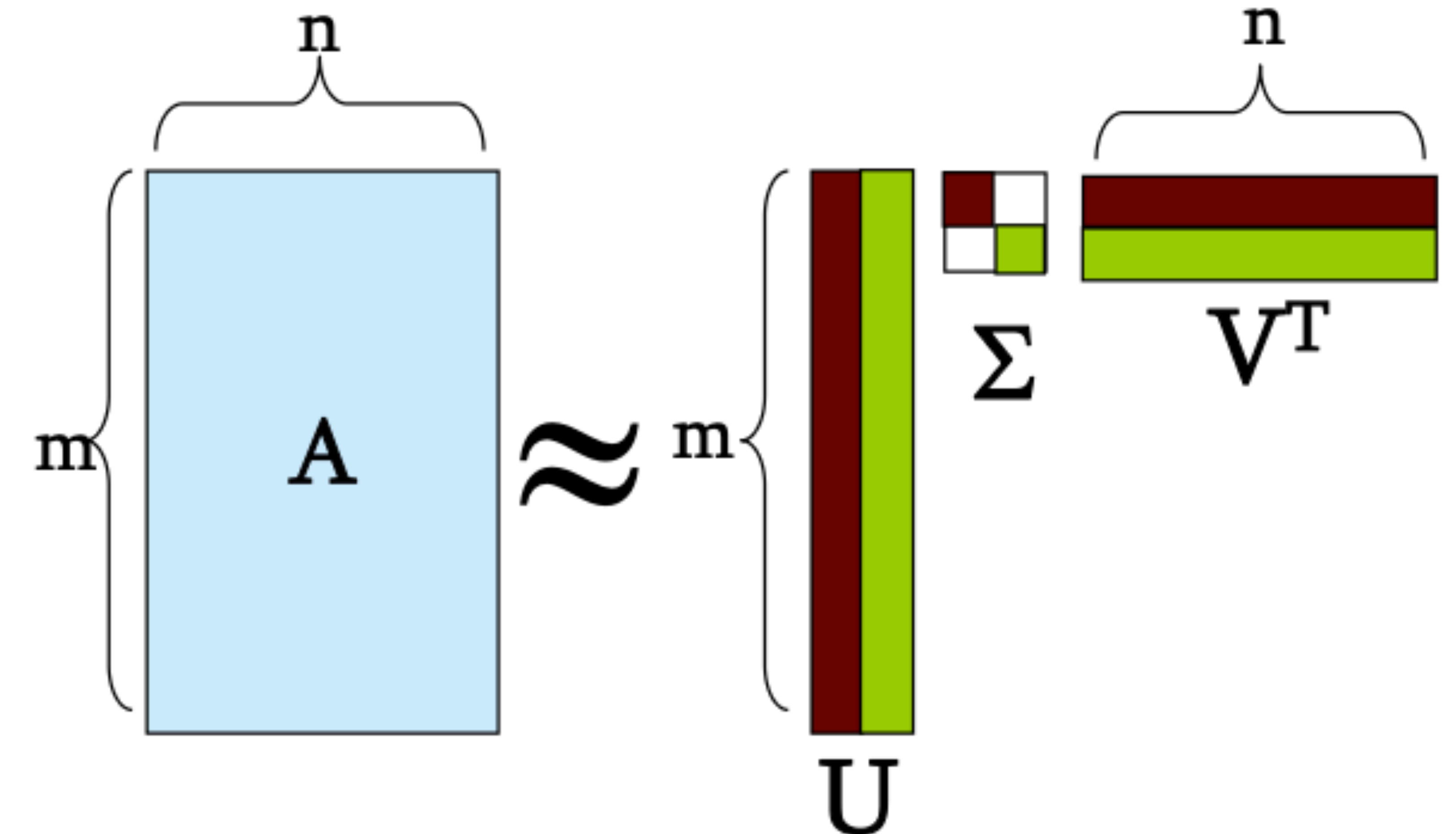
$$\ell = min_{P,Q} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

        - Where:
            - $\mathcal{K}$ is the set of (user, item) pairs **for which we have ratings**.
        - **Steps to optimize**:
            - Initialize P and Q using SVD
            - Gradient Descent:
                - $P \leftarrow P - \alpha \nabla_P$
                - $Q \leftarrow Q - \alpha \nabla_Q$
            - **Alternating Least Squares (ALS)**: This loss function is **non-convex.** Hence, we fix P and solve for Q (a linear problem), then fix Q and solve for P.

users (columns) / items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R



users (columns) / items (rows)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

R

≈

factors (columns) / items (rows)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

users (columns) / factors (rows)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

P^T

Recommender Systems

Further Improvements

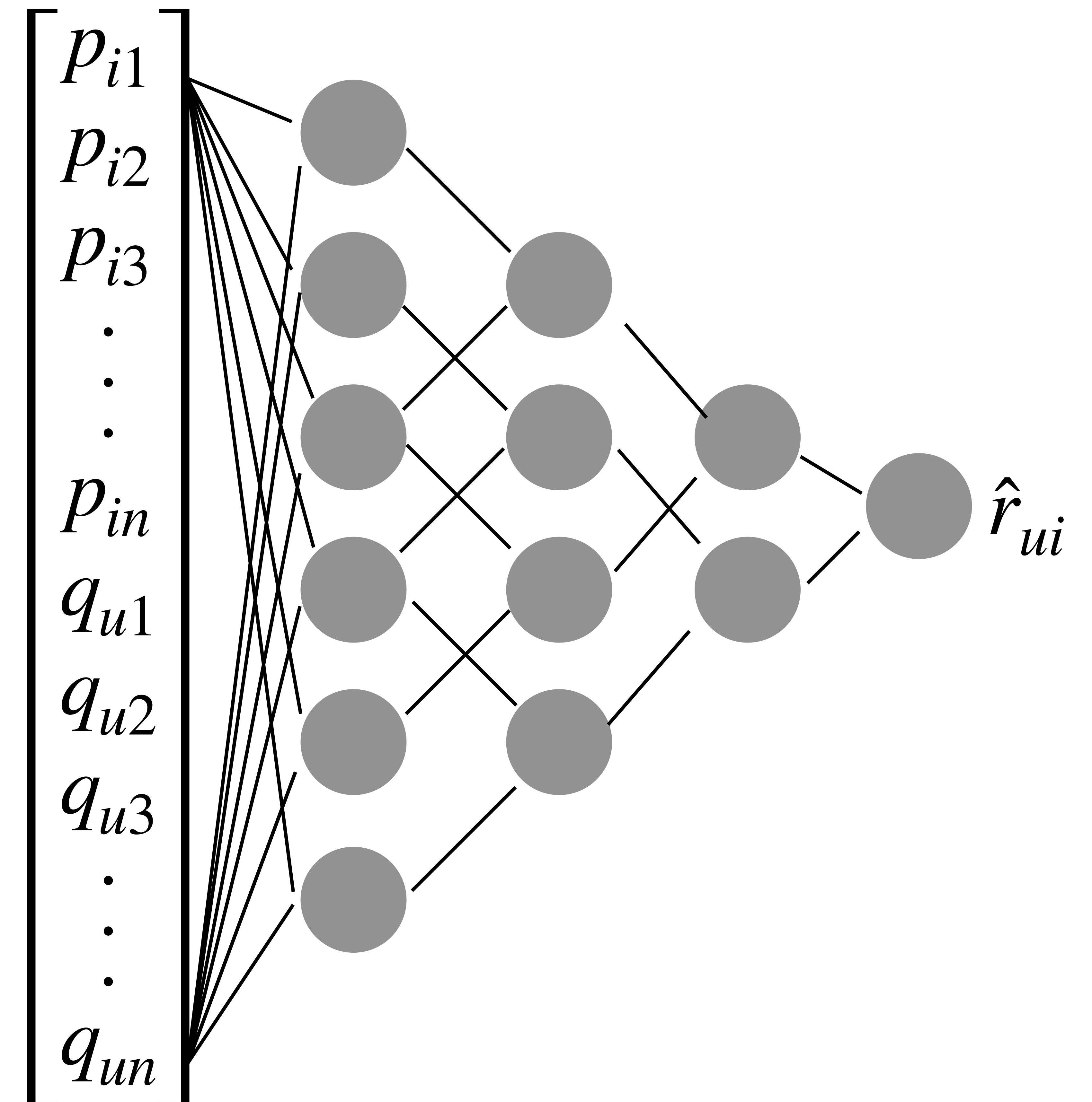
- We decompose the sparse matrix R into two lower-rank matrices:
  - $P \in \mathbb{R}^{m \times k}$ (User-Latent Factor matrix)
  - $Q \in \mathbb{R}^{n \times k}$ (Item-Latent Factor matrix)
- The predicted rating is the **dot product**

$$\hat{r}_{ui} = q_i^T p_u = \sum_{f=1}^{k} p_{uf} \cdot q_{if}$$

Replace dot product with an MLP

$$\hat{r}_{ui} = MLP([p_i, q_u])$$

Neural Collaborative Filtering



R (rows: items; columns: users)

users											
1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

Q (rows: items; columns: factors)

factors		
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

P^T (rows: factors; columns: users)

users										
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

Recommender Systems

Two Tower Models

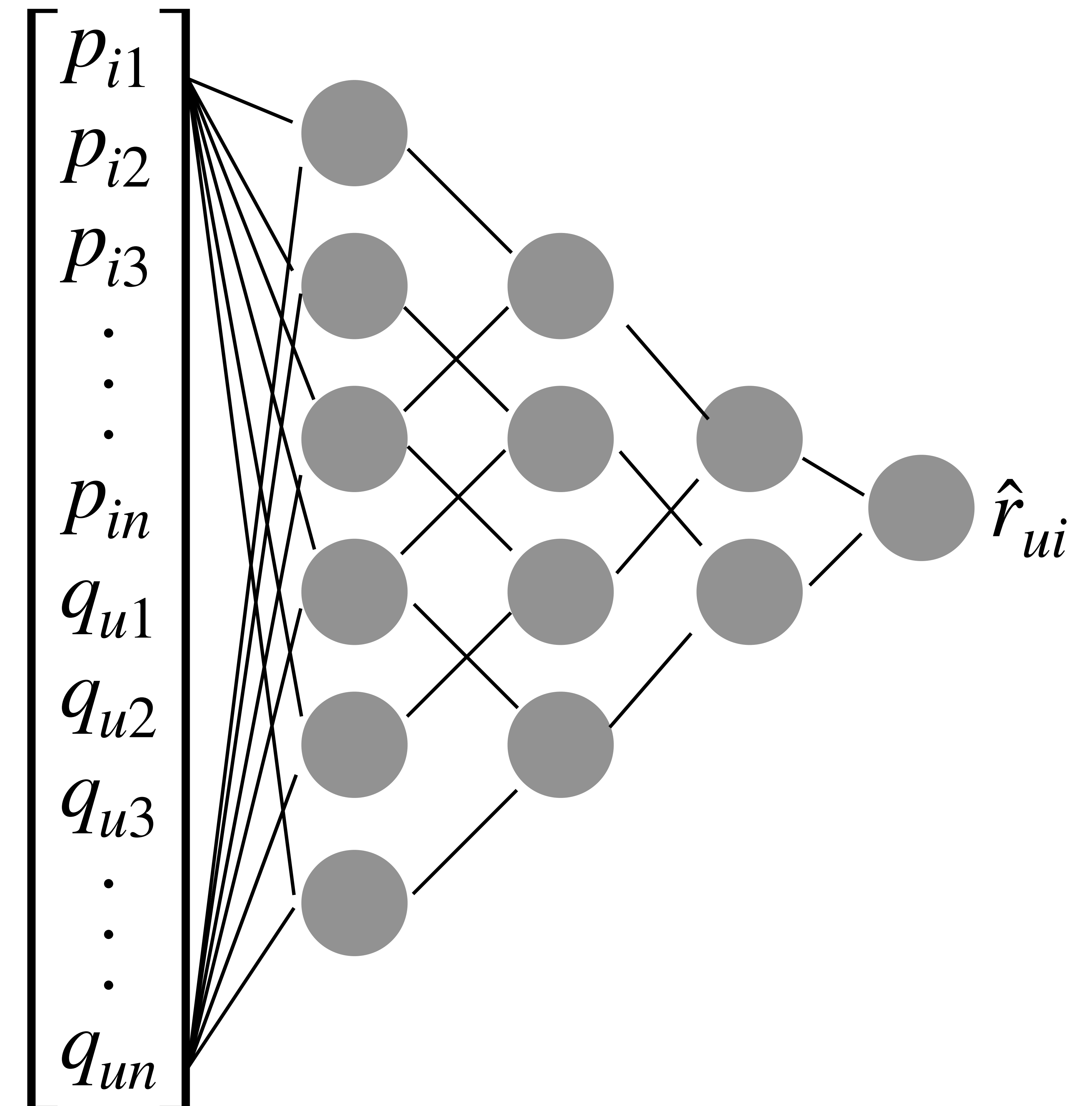
Current industry standard:

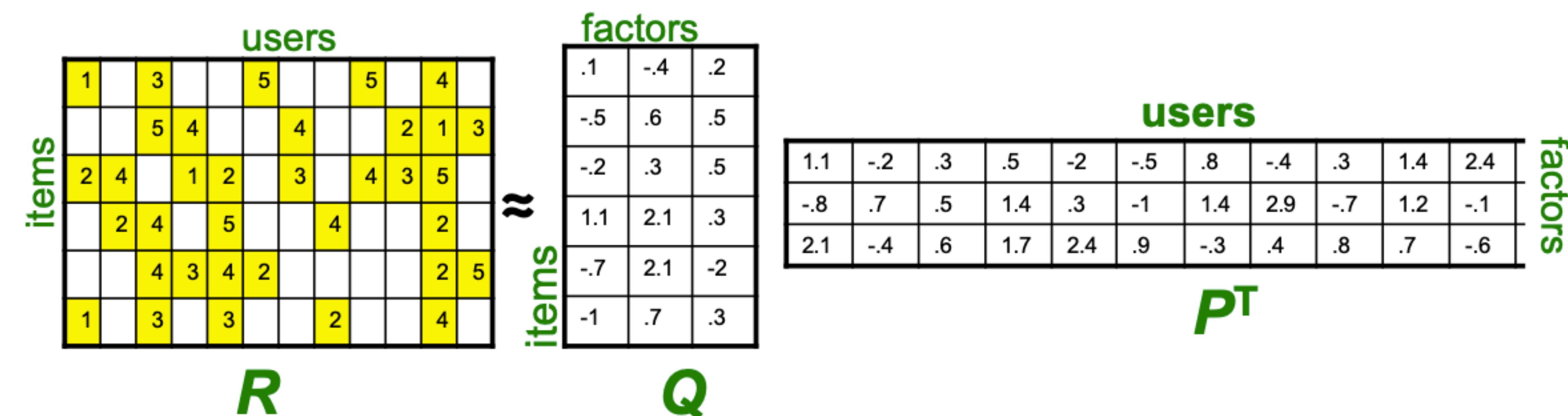
$$P_i = MLP(\text{item features}) \text{ (tower 1)}$$

$$Q_u = MLP(\text{user features}) \text{ (tower 2)}$$

$$\hat{r}_{ui} = MLP([p_i, q_u])$$

- **Why two towers?**
  - At serving time, you precompute all item embeddings p_i and store them in a vector index
  - For a new user request, compute q_u online, then do approximate nearest neighbor (ANN) search over all p_i
  - Retrieve top-k candidates in O(log n) time - scalable to billions of items





Recommender Systems

Cold Start Problems

- **New User Cold Start**:
  - A brand-new user has no interaction history. CF has nothing to go on.
  - **Solutions**: Ask for explicit preferences onboarding (rate 5 movies you've seen), use demographics for initial recommendations, recommend globally popular items, use content-based fallback
- **New Item Cold Start**:
  - A brand-new item has no ratings/interactions.
  - **Solutions**: Use content features (CBF works immediately), use item metadata for initialization of item embeddings, promote new items to a random subset of users to gather feedback

Recommender Systems

Ranking Metrics

- Did we show the user things they actually wanted?
- Precision@k:
  - Of the **top-k** items we recommended, what fraction did the user actually like?

$$Precision@k = \frac{|\text{relevant items in top-k}|}{k}$$

- Recall@k:
  - Of all items the user actually likes, what fraction did we include in our **top-k**?

$$Recall@k = \frac{|\text{relevant items in top-k}|}{|\text{all relevant items}|}$$

- NDCG@k (Normalized Discounted Cumulative Gain):
  - Rewards highly relevant items appearing earlier in the list - **position matters**:

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)}$$

Where rel_i is the **relevance** of the item at position i

Next Class

- Monday - April 13th - In Person Extra Credits Quiz
- Wednesday - April 15th - Last Class - Review