https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 8: Transformers

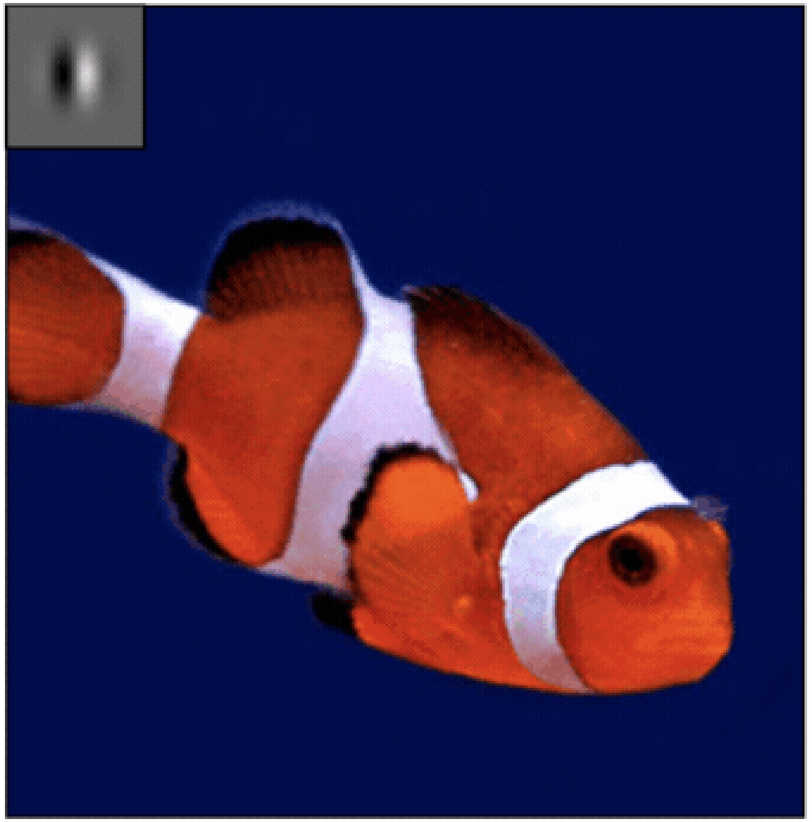
Shen Shen

April 5, 2024

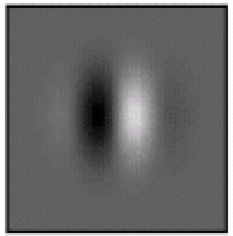
(many slides adapted from Phillip Isola and Kaiming He)

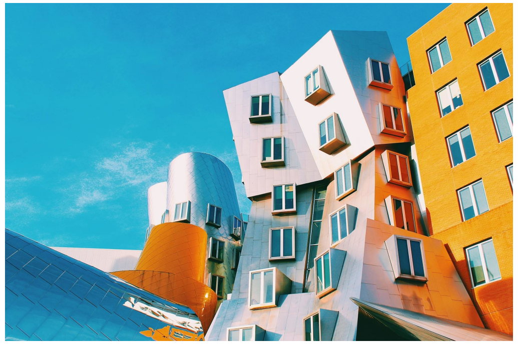
Outline

- Recap: CNN
- Transformers
  - Tokens
  - Attention
  - Self-attention
  - Learned Embedding
  - Full-stack
- (Applications and interpretation)



filter



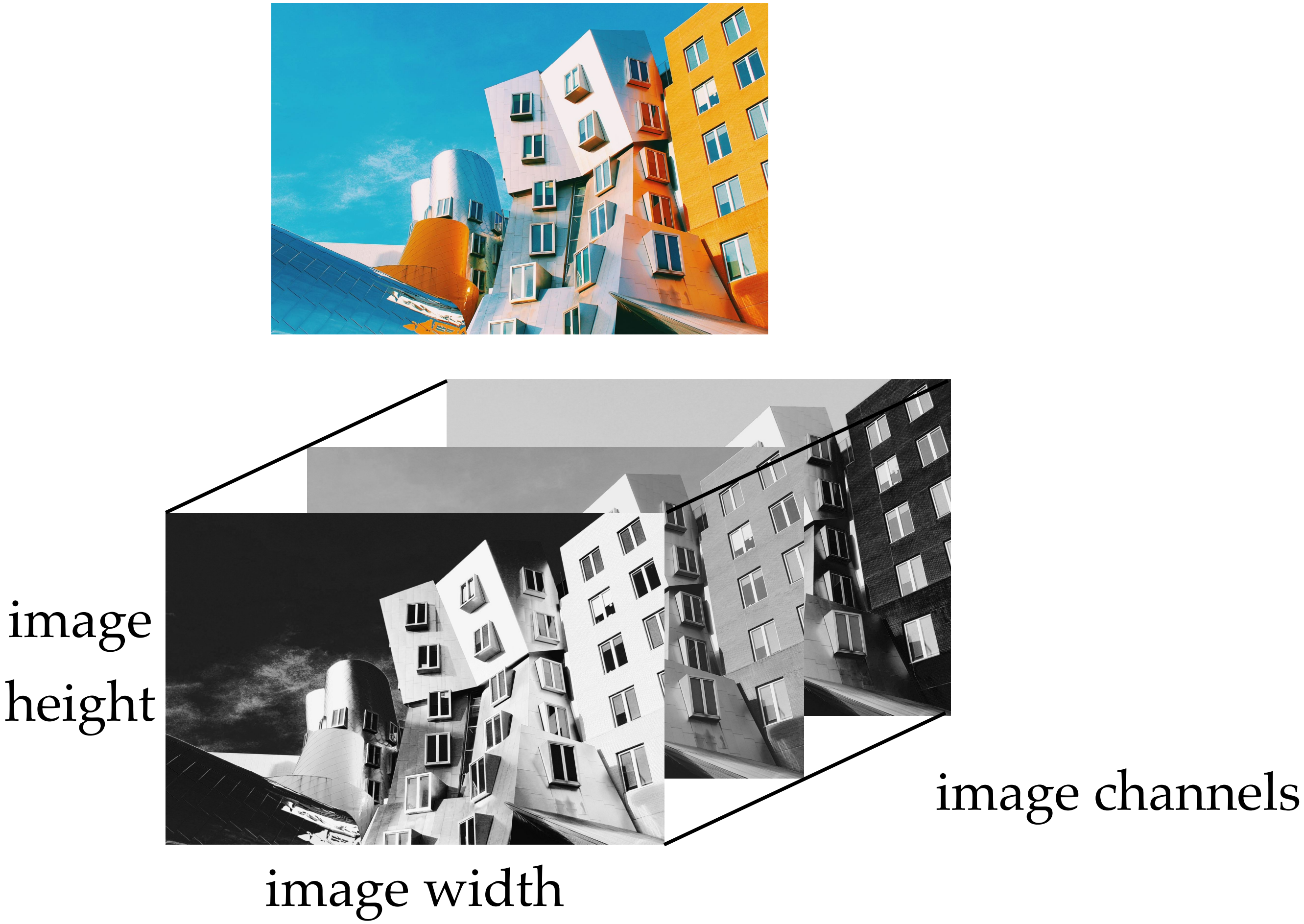


blue

green

red

[Photo by Zayn Shah on Unsplash]



[Photo by Zayn Shah on Unsplash]

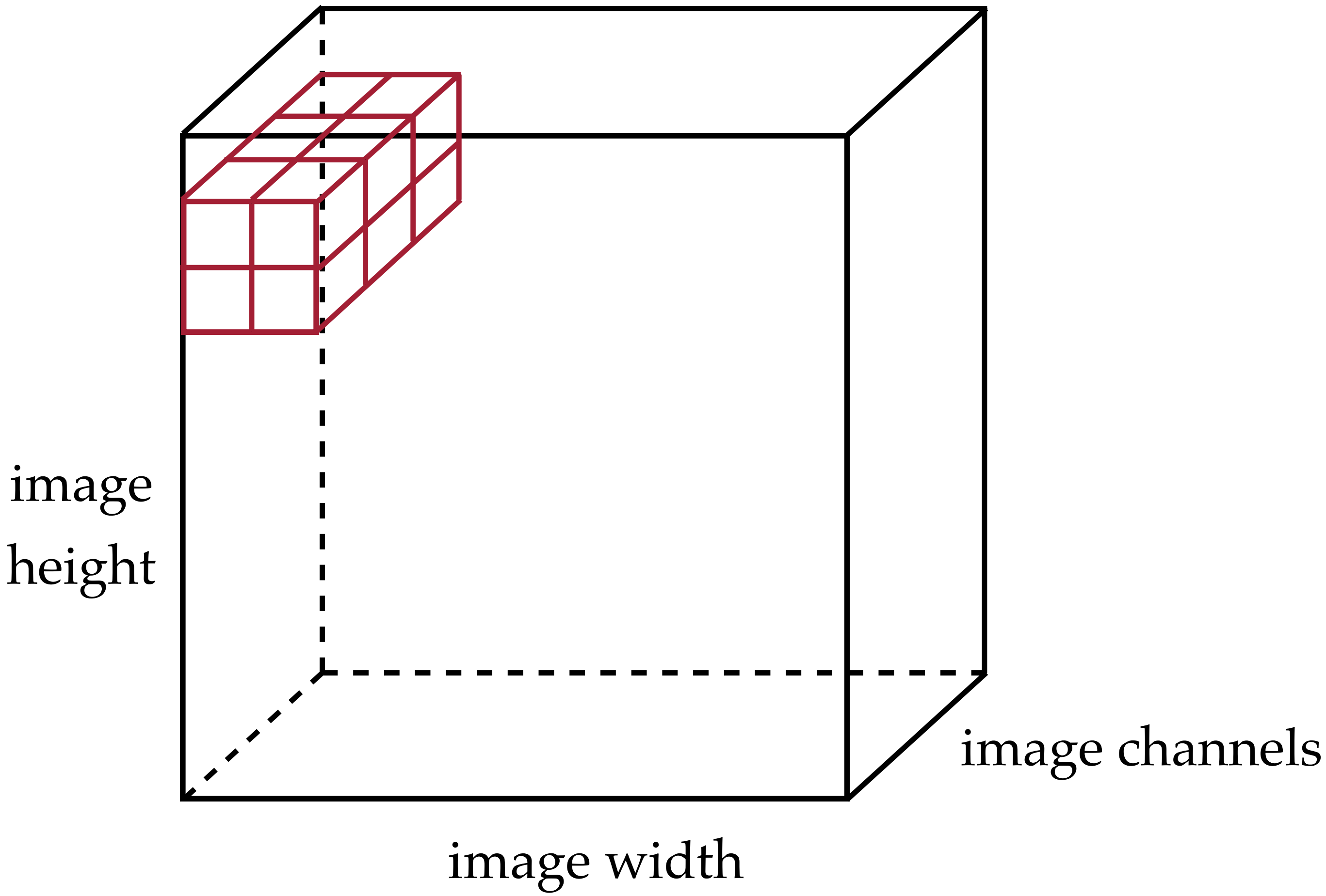
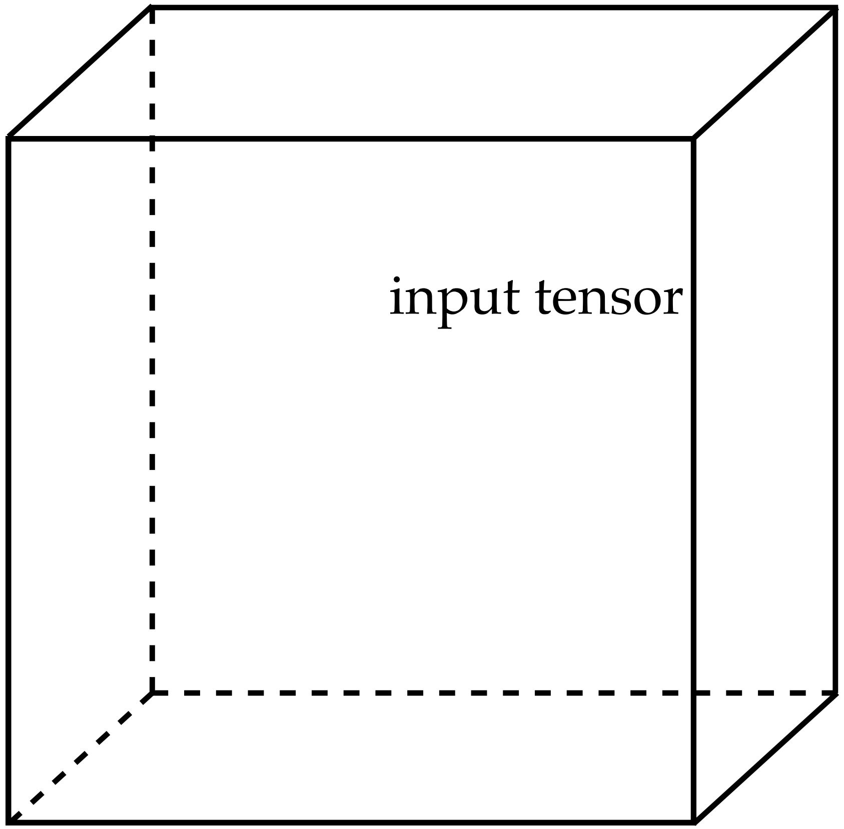
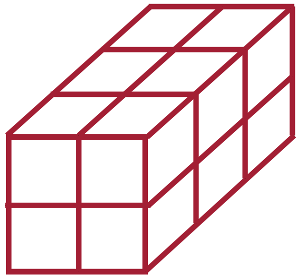


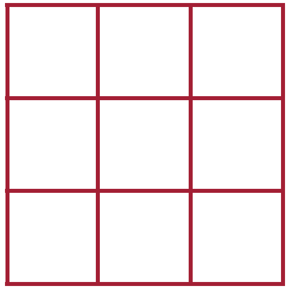
image
height
image channels
image width



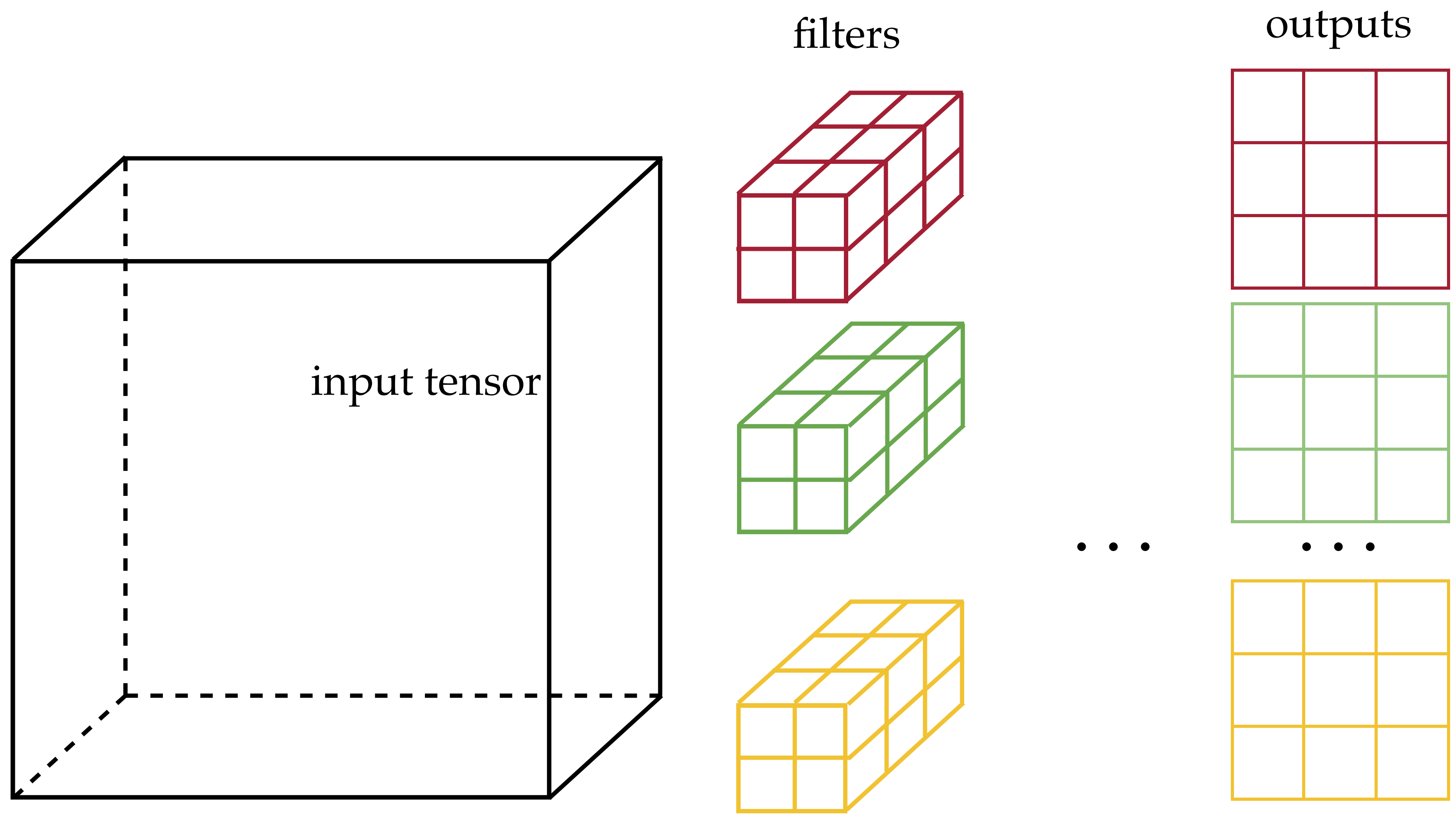
filter

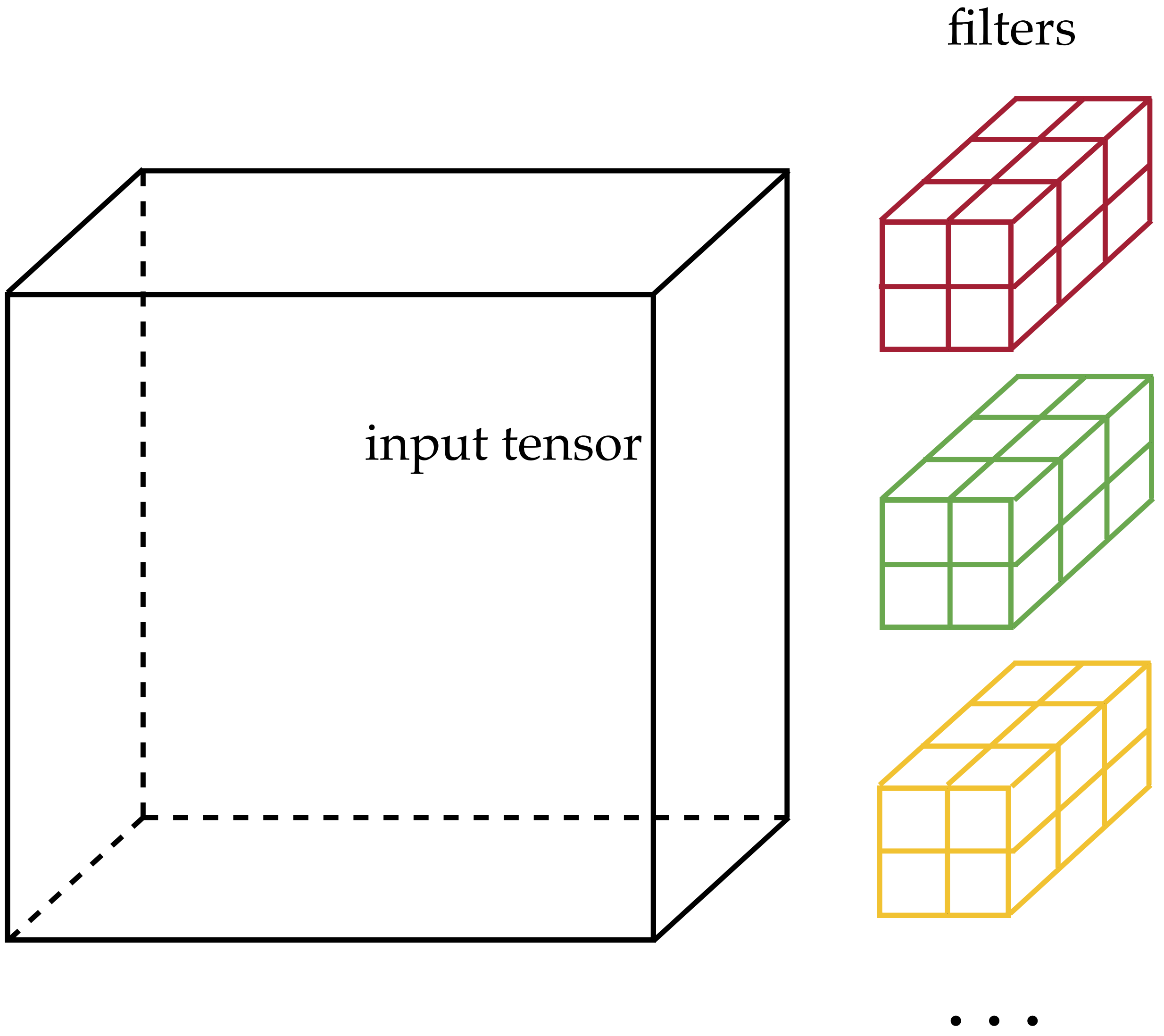


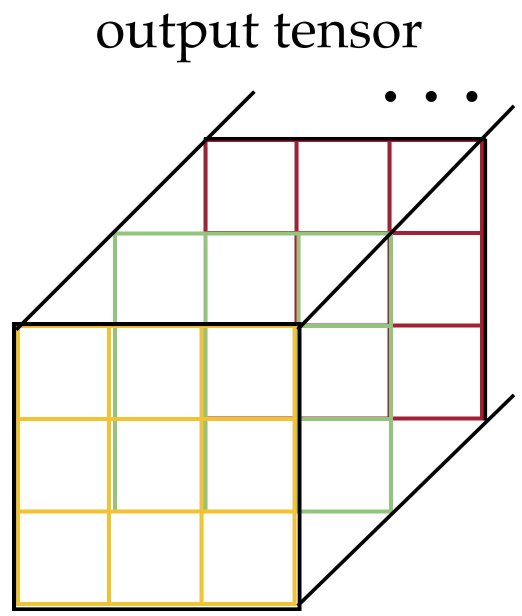
output



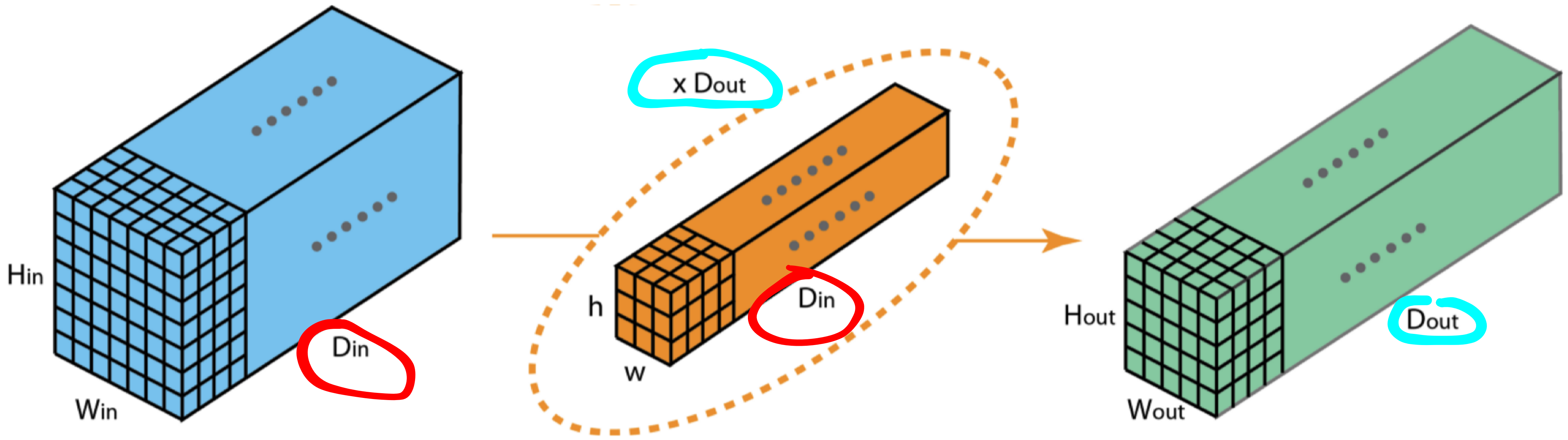
- 3d tensor input, depth d
- 3d tensor filter, depth d
- 2d tensor (matrix) output


filters
outputs
input tensor
. . .
. . .





- 3d tensor input, depth d
- k 3d filters:
  - each filter of depth d
  - each filter makes a 2d tensor (matrix) output
- total output 3d tensor, depth k



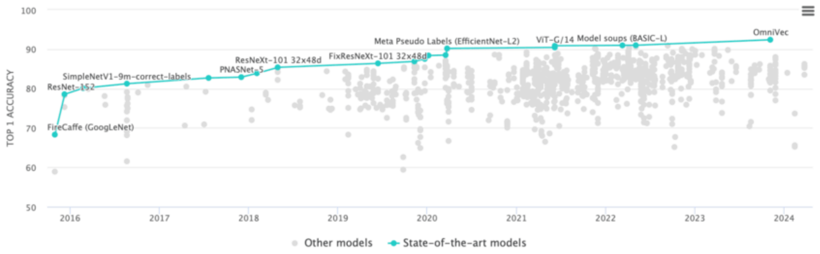
[image credit: medium]

Image Classification on ImageNet

Paperswithcode

Leaderboard | Dataset

View: Top 1 Accuracy by Date for All models



Filter: ImageNet-1k only | Transformer | ResNet | CNN | ImageNet-22k | EfficientNet | JFT-300M | MLP | ResNeXt | JFT-3B | Reversible | Neighborhood Attention | NAT Transformer | PatchConvnet | FPN | Conv+Transformer | ALIGN | MoE | Early Exit | Dynamic Model Arch | CNN+Transformer | CLIP data | SNN | IG-1B | Swin-Transformer | Teacher-22k | Vision Transformer | CrossCovarianceAttention | FLD-900M | Pure CNN | YFCC-15M | Laion-400M | Contrastive | ConvNeXt | Self-Supervised Learning | RegNet | Mixer | Memory-Centric | CLIP Pre-trained | untagged | Hardware Burden | Operations per network pass | Robustness reports

Edit Leaderboard

Model	Top 1 Accuracy	Number of params	GFLOPs	energy consumption	Extra Training Data	Paper	Code	Result	Year	Tags
iVec	92.4%				×	OmniVec: Learning robust representations with cross modal sharing			2023	
C-L , fine-tuned)	91.1%	2440M			×					Conv+Transformer ALIGN JFT-3B
uned)	91.0%	2100M			×	CoCa: Contrastive Captioners are Image-Text Foundation Models			2022	ALIGN Transformer JFT-3B
el soups IC-L)	90.98%	2440M			×	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	ALIGN JFT-3B Conv+Transformer
el soups G/14)	90.94%	1843M			×	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	JFT-3B Transformer
	90.9%	3900M			×	PaLI: A Jointly-Scaled Multilingual Language-Image Model			2022	Transformer JFT-3B

cont'd

Model	Top 1 Accuracy	Number of params	GFLOPs	Extra Training Data	Paper	Code	Result	Year	Tags
:Net-7	90.88%	2440M	2586	×	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
G/14)	90.71%	1843M		×					Transformer JFT-3B
en)	90.60%	2100M		×	CoCa: Contrastive Captioners are Image-Text Foundation Models			2022	Transformer ALIGN JFT-3B
:Net-6	90.45%	1470M	1521	×	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
G/14	90.45%	1843M	2859.9	×	Scaling Vision Transformers			2021	Transformer
T-G	90.4%	1437M	1038	×	DaViT: Dual Attention Vision Transformers			2022	Transformer
T-H	90.2%	362M	334	×	DaViT: Dual Attention Vision Transformers			2022	Transformer
Pseudo Labels ientNet-L2)	90.2%	480M		×	Meta Pseudo Labels			2020	JFT-300M EfficientNet
V2-G	90.17%	3000M		×	Swin Transformer V2: Scaling Up Capacity and Resolution			2021	Transformer
nImage-DCNv3-G Pre-training)	90.1%	3000M		×	InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions			2022	
VS 5.5B)	90.1%	6500M		×	The effectiveness of MAE pre-pretraining for billion-scale pretraining			2023	
nce-CoSwin-H	90.05%	893M		×	Florence: A New Foundation Model for Computer Vision			2021	FLD-900M Transformer
Pseudo Labels ientNet-B6-Wide)	90%	390M		×	Meta Pseudo Labels			2020	JFT-300M EfficientNet
:ol-H	90.0%	2158M		×	Reversible Column Networks			2022	Pure CNN Reversible CNN

Lessons from CNNs

Enduring principles:

1. Chop up signal into patches (divide and conquer)
2. Process each patch **identically** (and in **parallel**)

Transformers

Enduring principles:

1. Chop up signal into patches (divide and conquer)
2. Process each patch **identically** (and in **parallel**)

Follow the same principles:

1. via tokenization
2. via attention mechanism

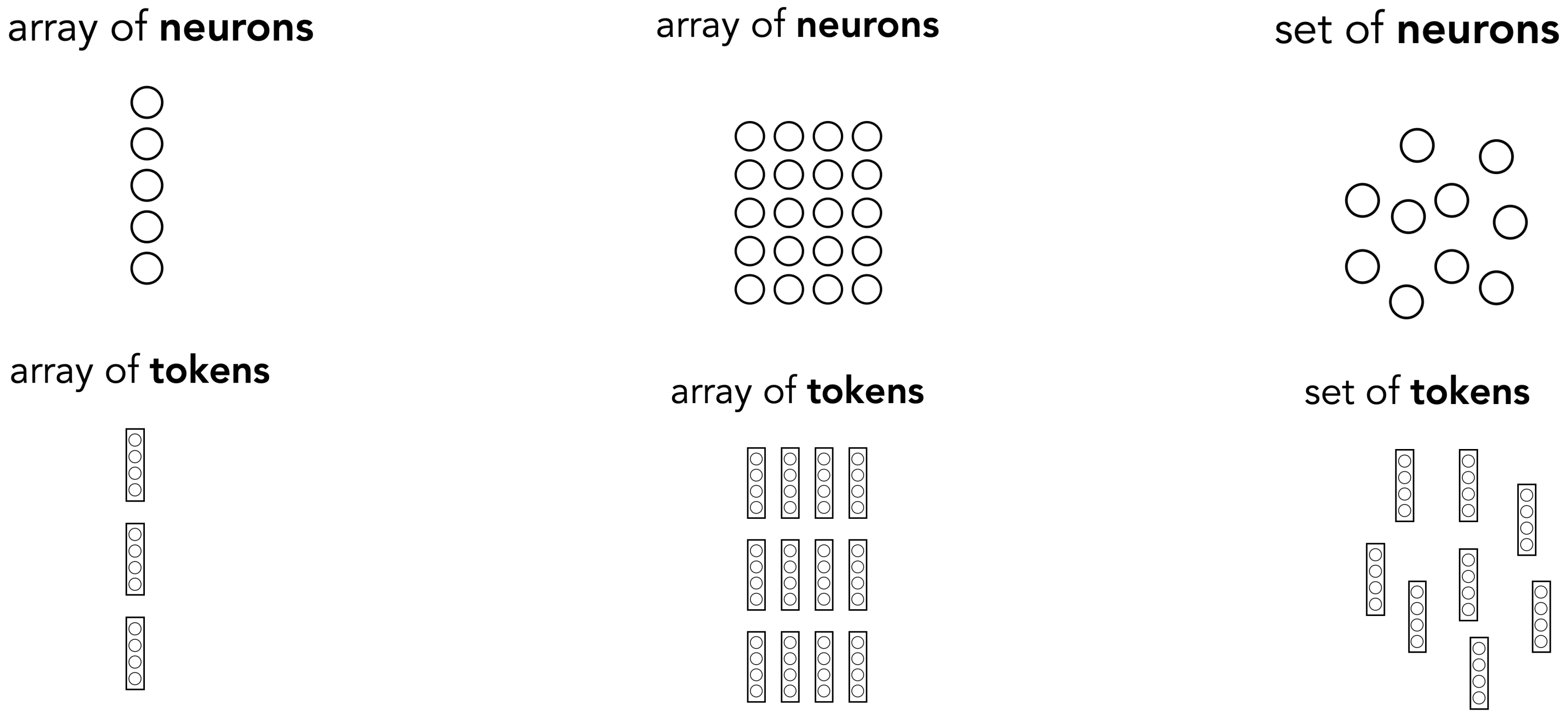
(conceptually: transformers are CNNs where the filter weights -- or here the attention -- dynamically change depending on the patch)

Outline

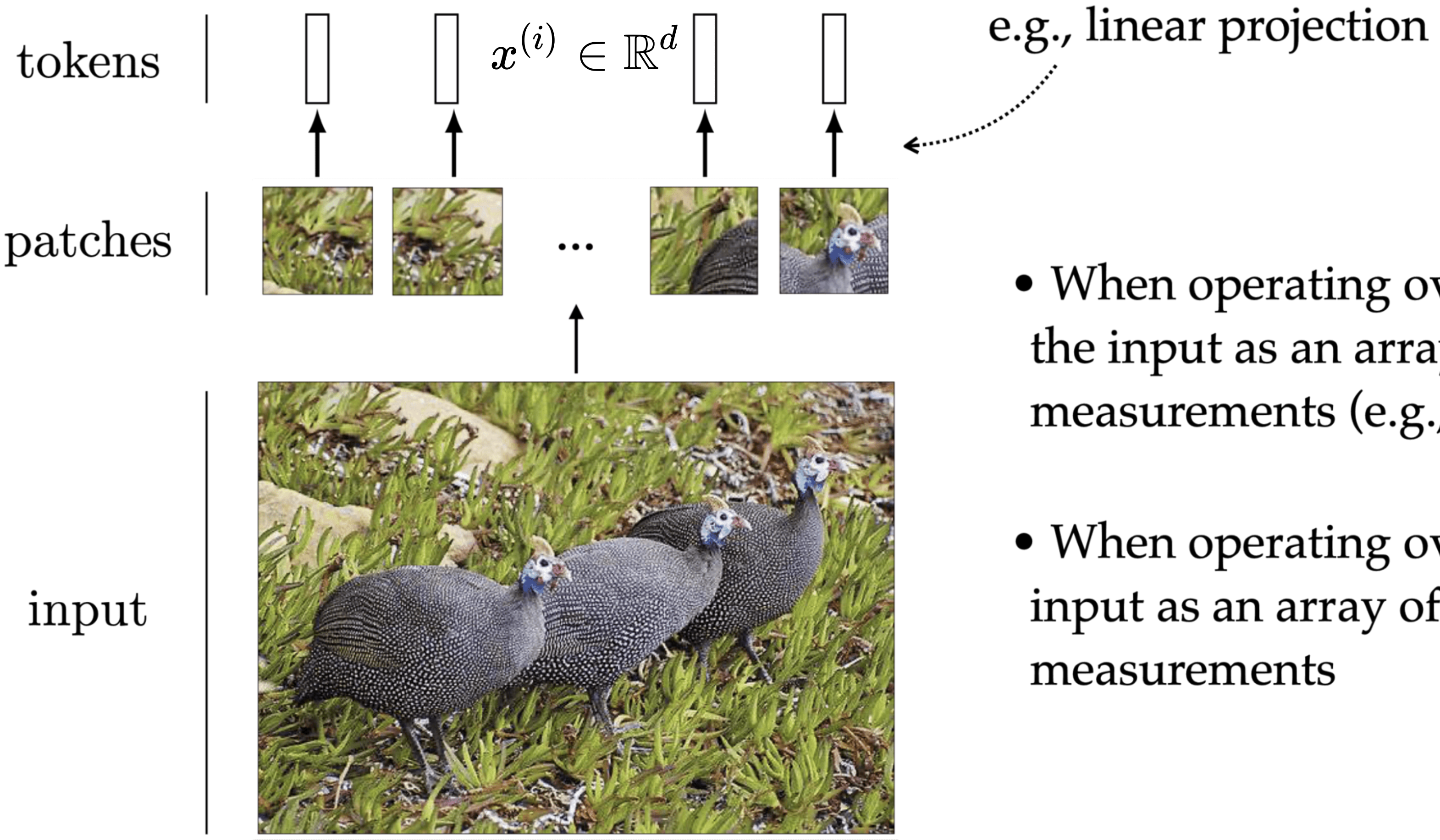
- Recap: CNN
- Transformers
  - Tokens
  - Attention
  - Self-attention
  - Learned Embedding
  - Full-stack
- (Applications and interpretation)

Tokens

- A token is just **transformer** lingo for a vector of neurons
- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons



Tokenizing the input data

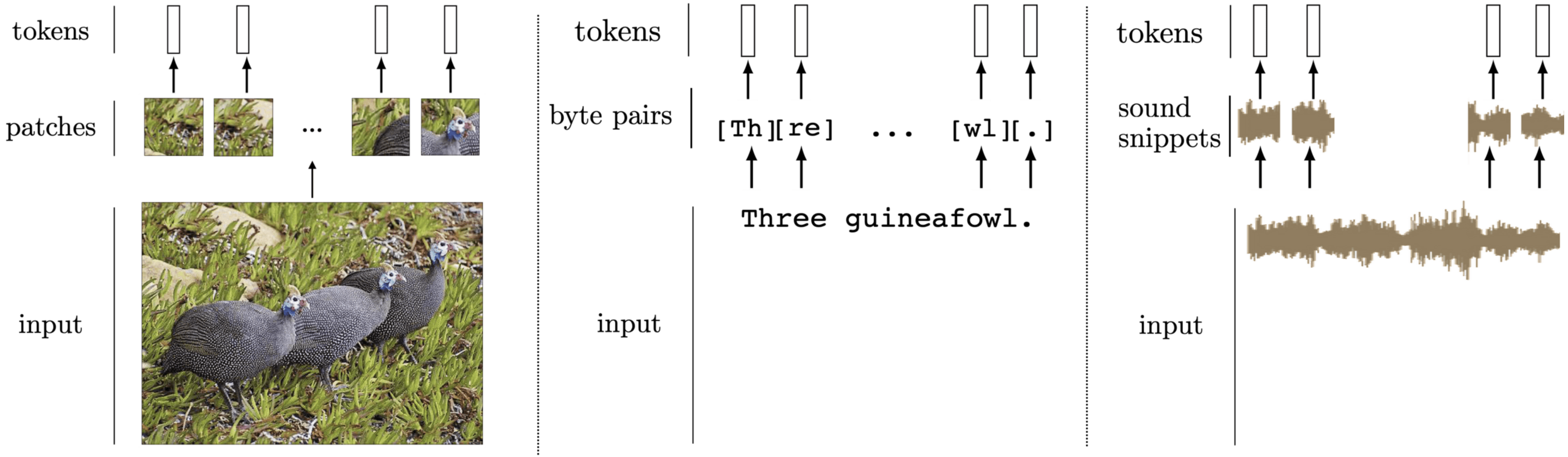


- When operating over *neurons*, we represent the input as an array of scalar-valued measurements (e.g., pixels)

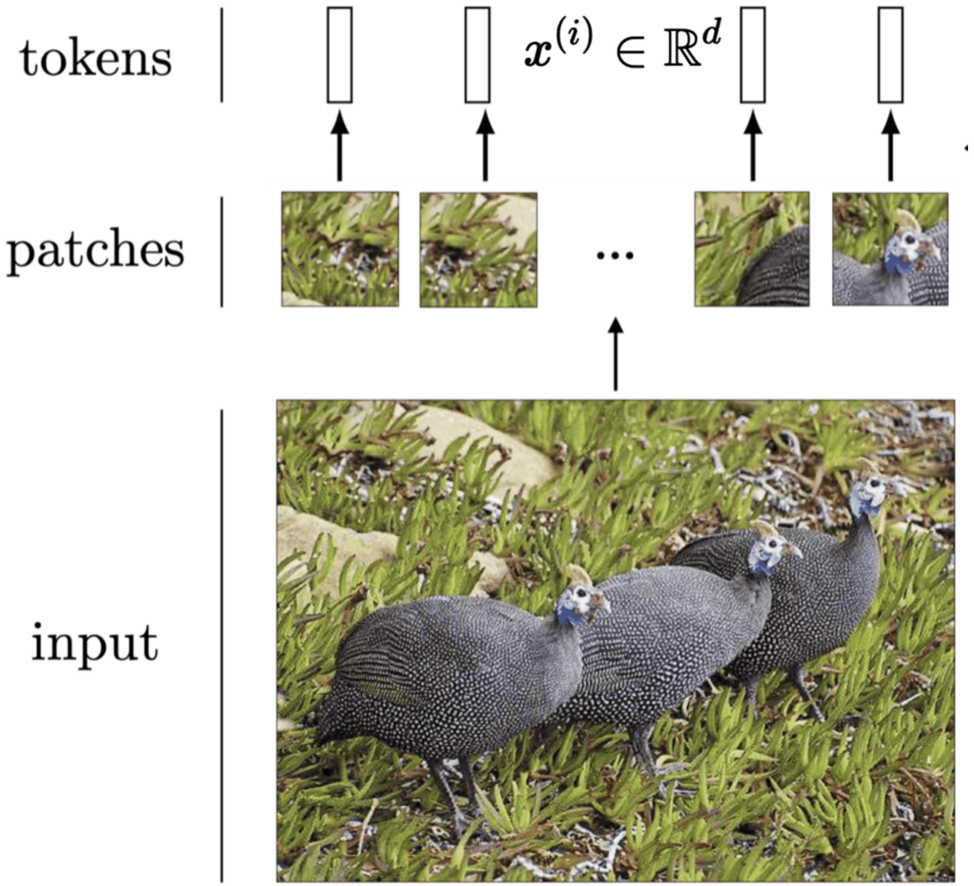
- When operating over *tokens*, we represent the input as an array of vector-valued measurements

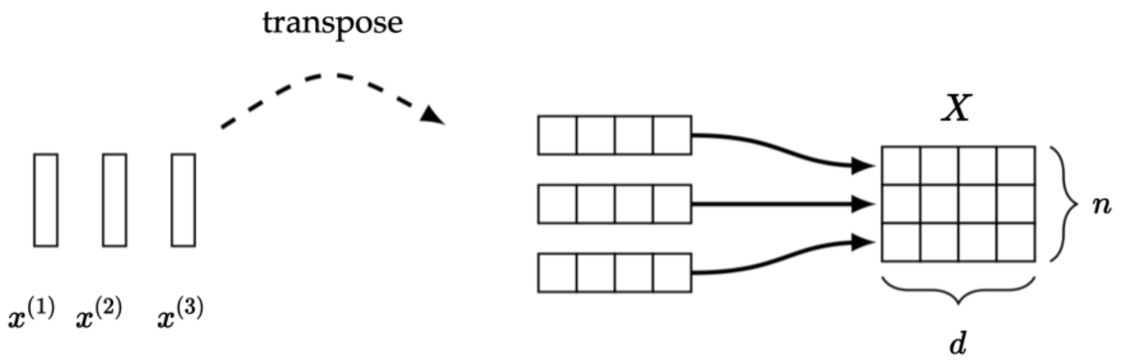
Tokenizing the input data

You can tokenize anything.
General strategy: chop the input up into chunks, project each chunk to a vector.



Token notations





- d is the size of each token ($x^{(i)} \in \mathbb{R}^d$)
- n is the number of tokens

Outline

- Recap: CNN
- Transformers
  - Tokens
  - Attention
  - Self-attention
  - Learned Embedding
  - Full-stack
- (Applications and interpretation)

Attention mechanism

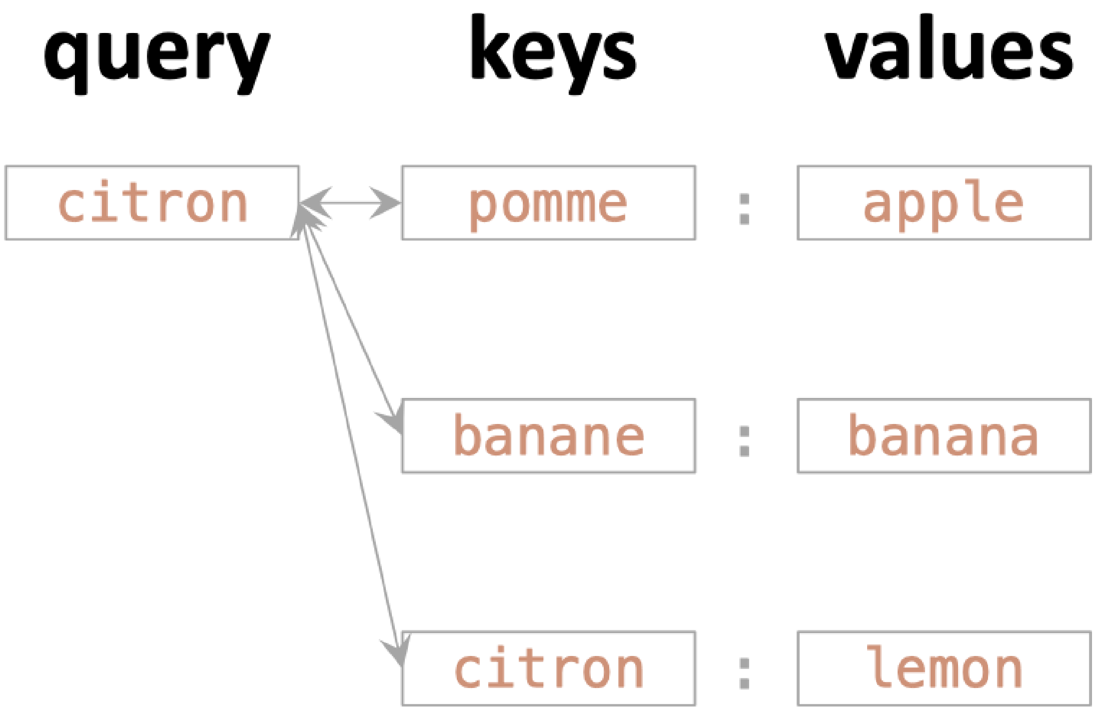
Let's start by thinking about dictionary look up

```
dict_fr2en = {
"pomme": "apple",
"banane": "banana",
"citron": "lemon"
}
```

keys		values
pomme	:	apple
banane	:	banana
citron	:	lemon

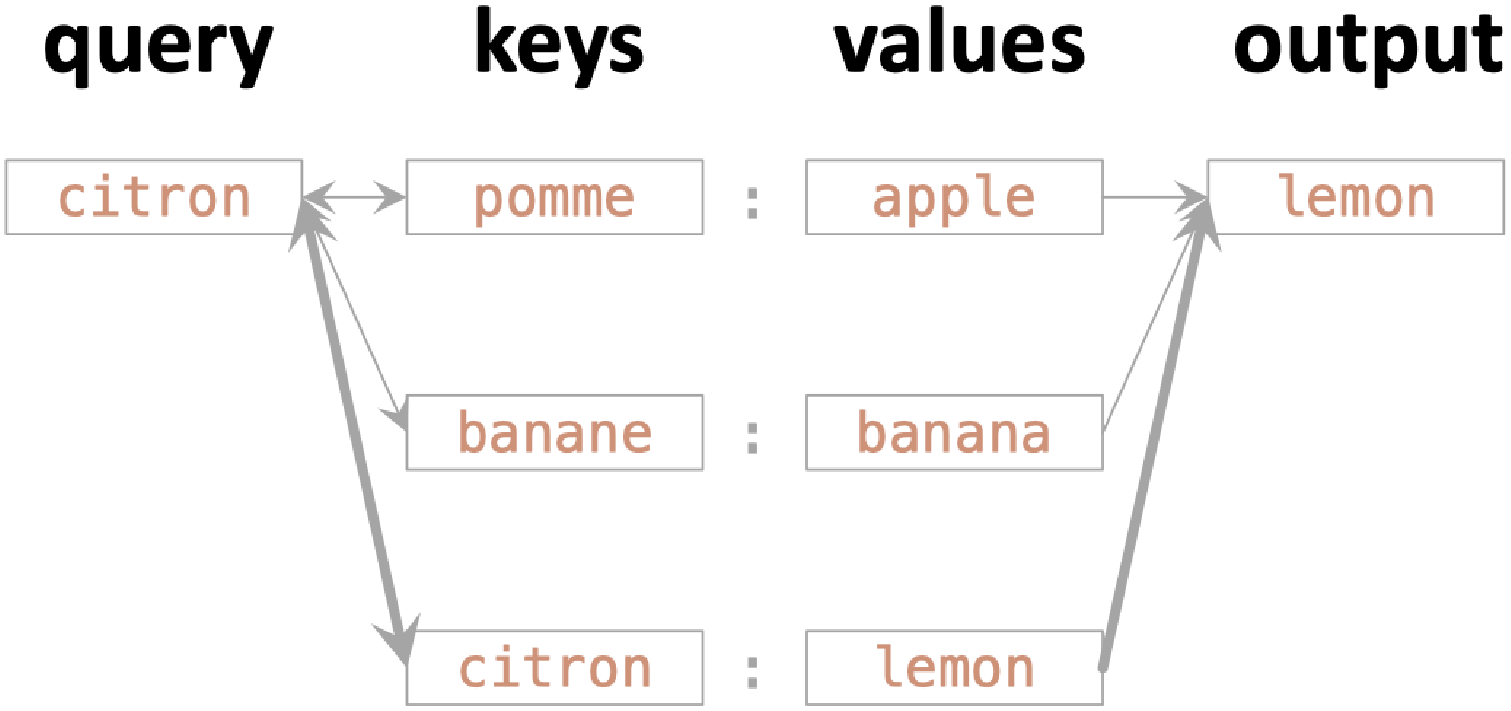
```python
dict_fr2en = {
"pomme": "apple",
"banane": "banana",
"citron": "lemon"
}

query = "citron"
output = dict_fr2en[query]
```



```
dict_fr2en = {
"pomme": "apple",
"banane": "banana",
"citron": "lemon"
}

query = "citron"
output = dict_fr2en[query]
```



```python
dict_fr2en = {
"pomme": "apple",
"banane": "banana",
"citron": "lemon"
}
```

What if we'd like to run

```python
query = "orange"
output = dict_fr2en[query]
```

Python would complain.

But you might see the rationale of:

```python
output = 0.8 * "lemon" + 0.1 * "apple"
+ 0.1 * "banana"
```

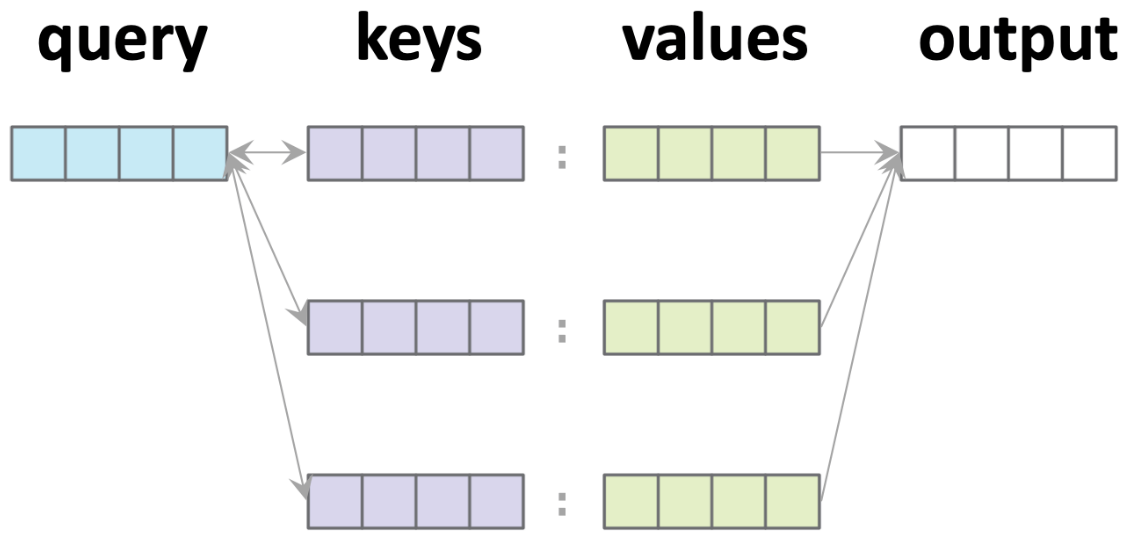
(though python would still complain)

- Why did the weights $[0.8, 0.1, 0.1]$ make sense?

  "soft" look up.

  Actually one way of understanding "**attention**"

- Can we generalize the thought process somewhat?



Sensible "abstraction / **embedding**"

Attention

Single-query example:

1. Similarity score w/ key j:

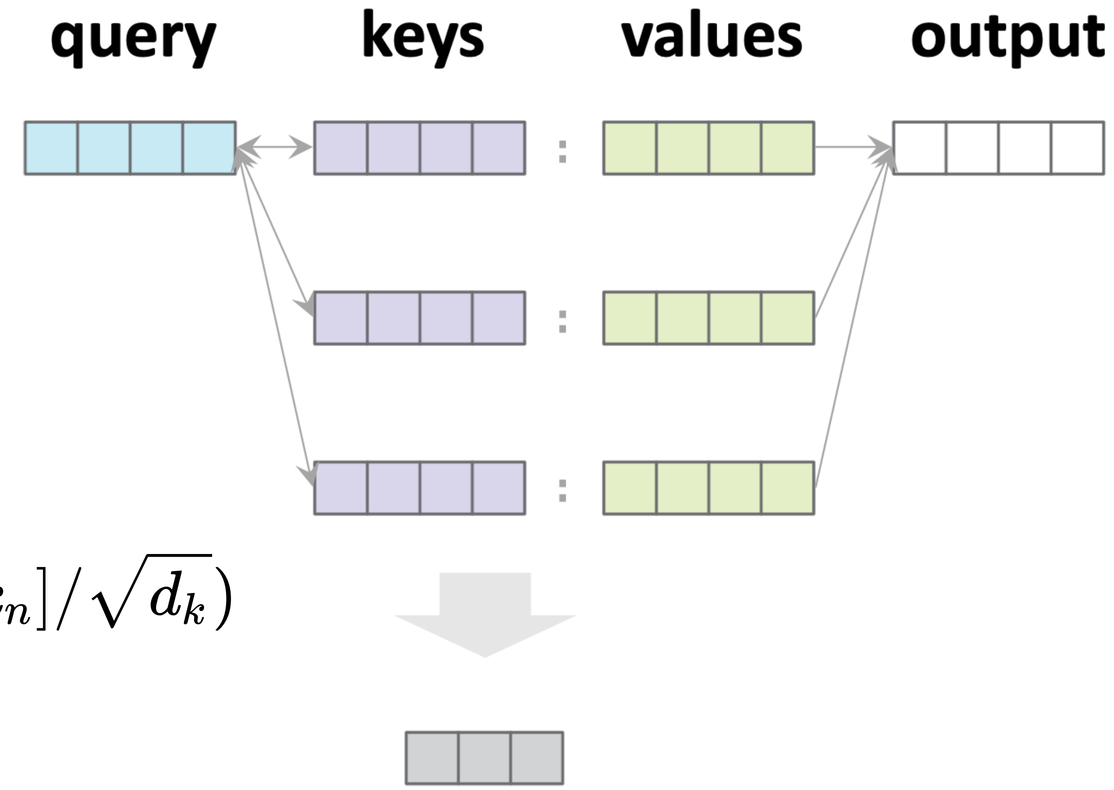
$$s_j = (q^T k_j) \ / \sqrt{d_k}$$

2. Attention weights (softmax'd scores):

$$\begin{aligned} a &= \mathrm{softmax}([q^\top k_1, q^\top k_2, q^\top k_3, \ldots, q^\top k_n]/\sqrt{d_k}) \\ &= \mathrm{softmax}([s_1, s_2, s_3, \ldots, s_n]) \\ &= [e^{s_1}, e^{s_2}, \ldots, e^{s_n}]/\Sigma_j e^{s_j} \end{aligned}$$

3. Output: attention-weighted sum:

$$y = \sum_j a_j v_j$$



- n: number of keys
- d_q: dim(query embedding)
- d_k: dim(key embedding)
- d_v: dim(value embedding)

Multi-query example:

1. Similarity score of (query i and key j):

$$s_{ij} = ({q_i}^T k_j) \,/\, \sqrt{d_k}$$

2. Attention weights (softmax'd scores):

For each query i,

$$a_i = \mathrm{softmax}([s_{i1}, s_{i2}, s_{i3}, \;\ldots, s_{in_k}])$$

Stack all such a_i vertically

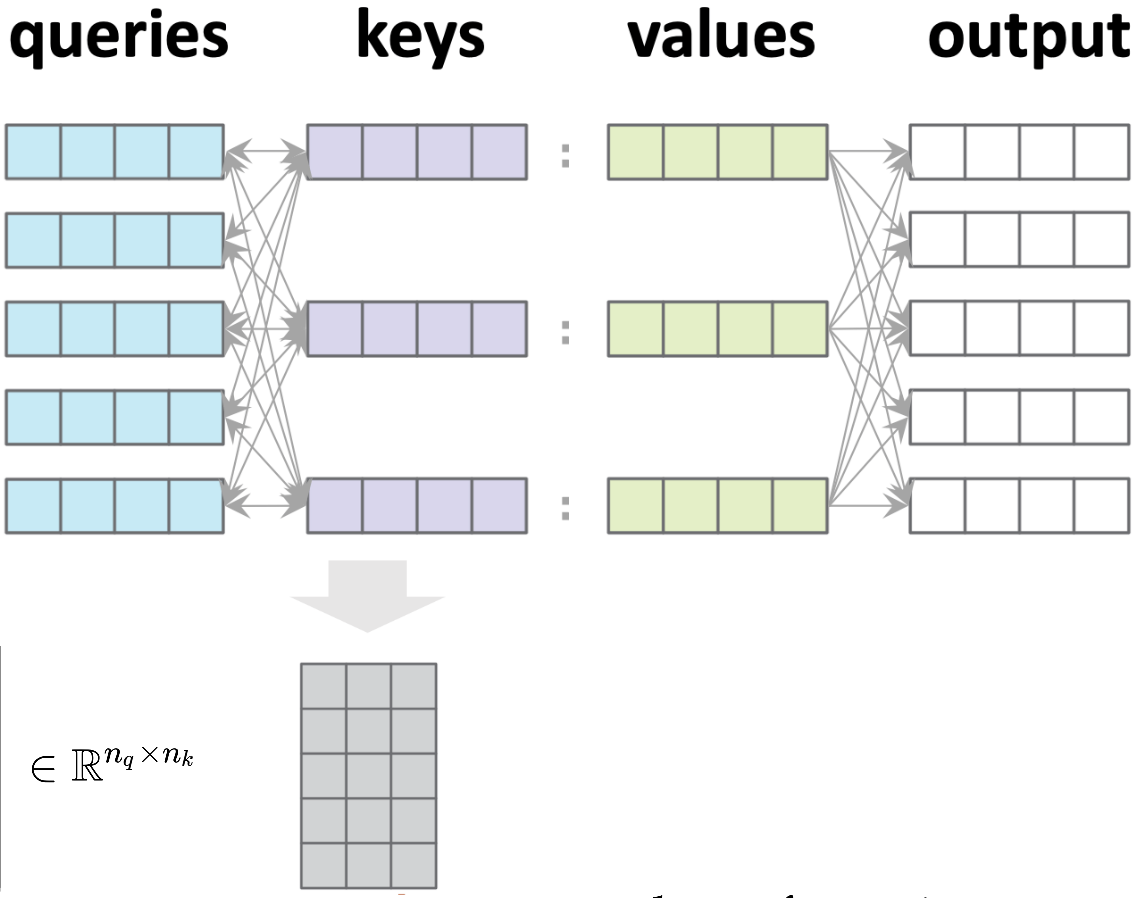
$$A = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nq} \end{bmatrix} \in \mathbb{R}^{n_q \times n_k}$$

3. Output: attention-weighted sum:

For each query i, $y_i = \sum_j a_{ij} v_j$

Stack all such y_i vertically

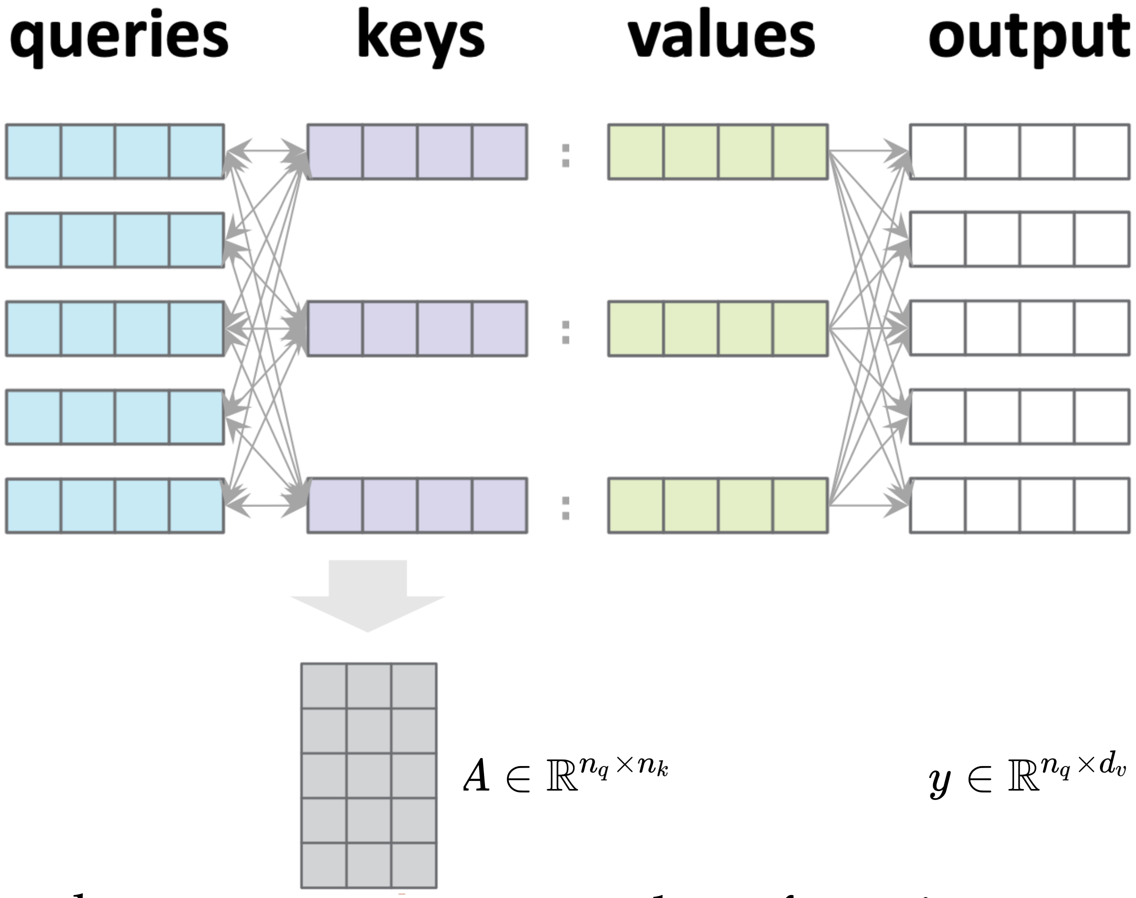
$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{nq} \end{bmatrix} \in \mathbb{R}^{n_q \times d_v}$$



- n_q: number of queries
- n_k: number of keys
- d_q: dim(query embedding)
- d_k: dim(key embedding)
- d_v: dim(value embedding)

Comments:

- Attention says nothing about how to get queries/keys/values.
- Attention itself is **parameter-free.**
- Shapewise, we only need:
  - $d_k = d_q$ (so we often omit d_q)
  - any other shapes need not match:
    - n_q need not equal n_k
    - d_v need not equal d_k
- Note all queries are processed in **parallel.**
  - **No** cross-wiring between queries.
  - **Any** output is connected to **every** value and **every** key, but **only** its corresponding query.
- This is the vanilla default attention mechanism, aka, "query-key-value dot-product cross attention".
- One such attention mechanism is called one "Head"

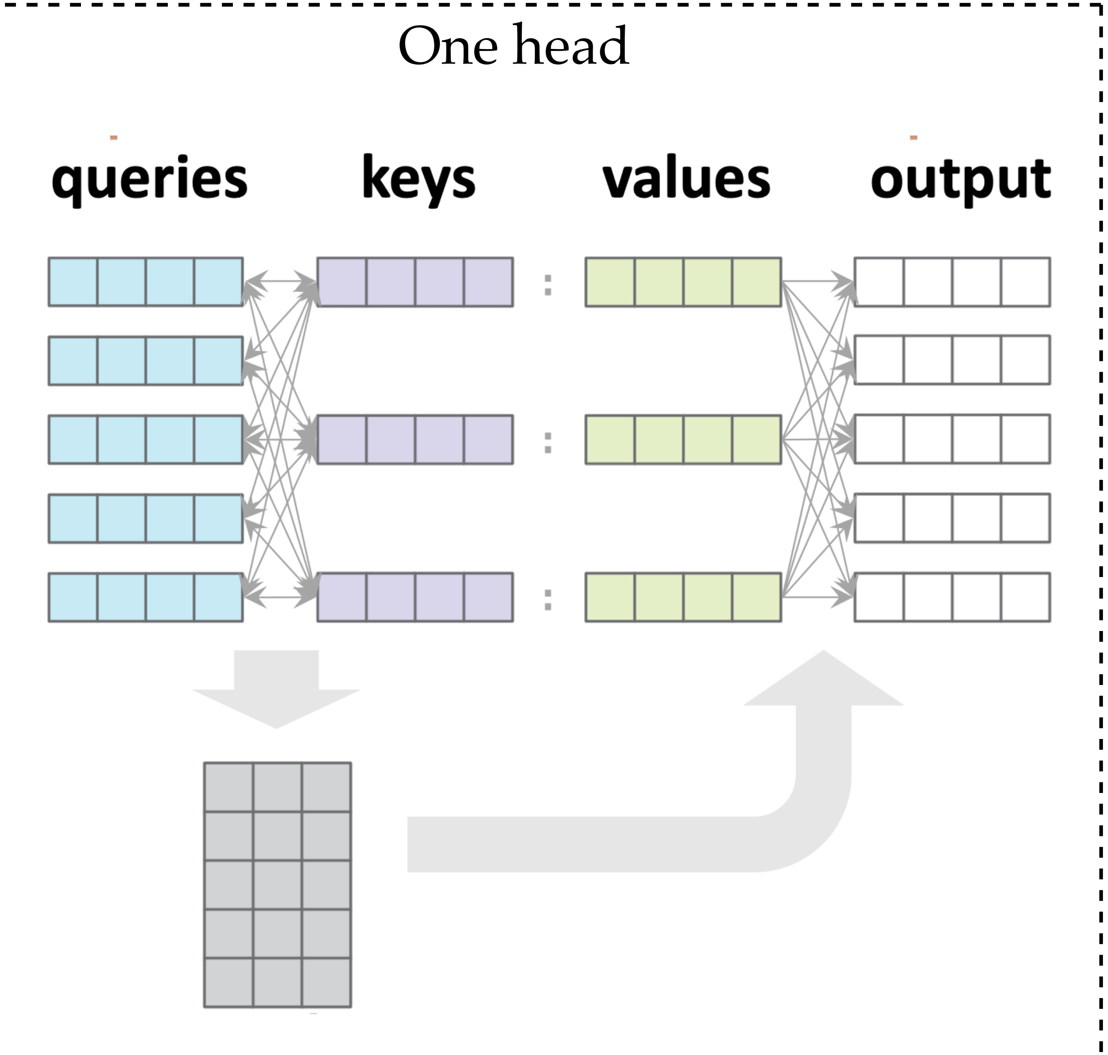


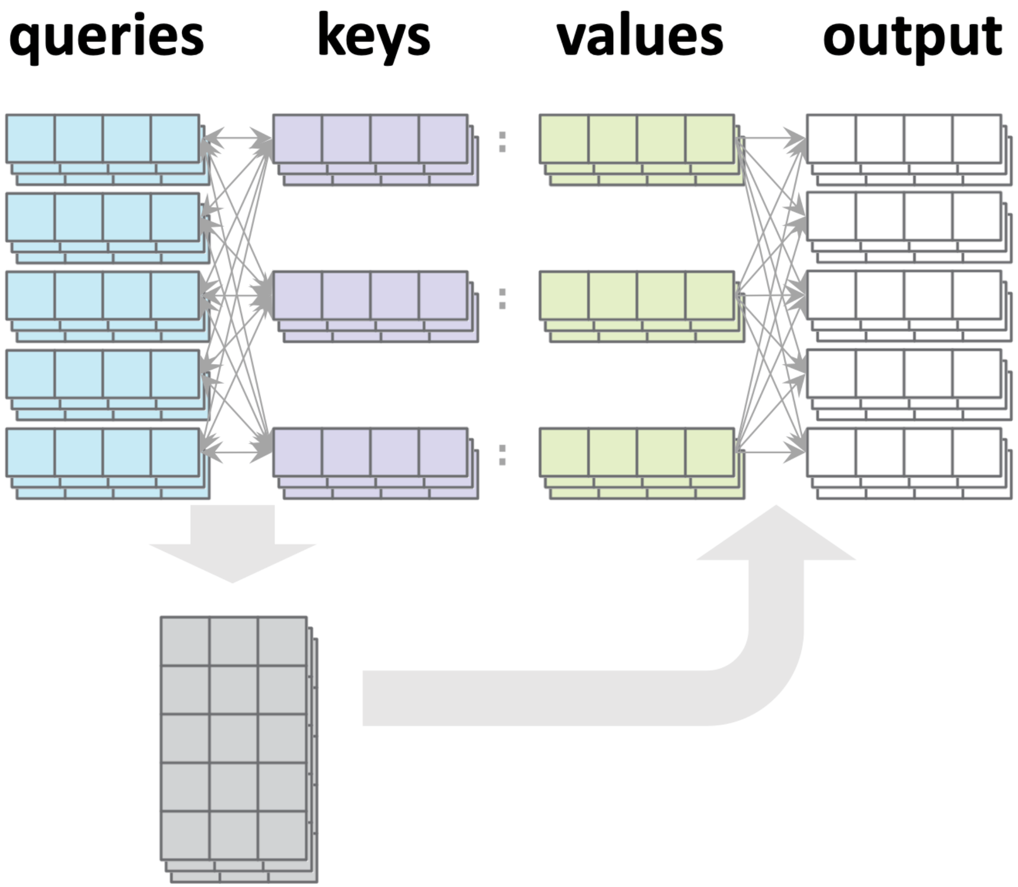
- n_q: number of queries
- n_k: number of keys
- d_q: dim(query embedding)
- d_k: dim(key embedding)
- d_v: dim(value embedding)

Multi-head Attention

Rather than having just one way of attending, why not have multiple?

Repeat in **parallel**



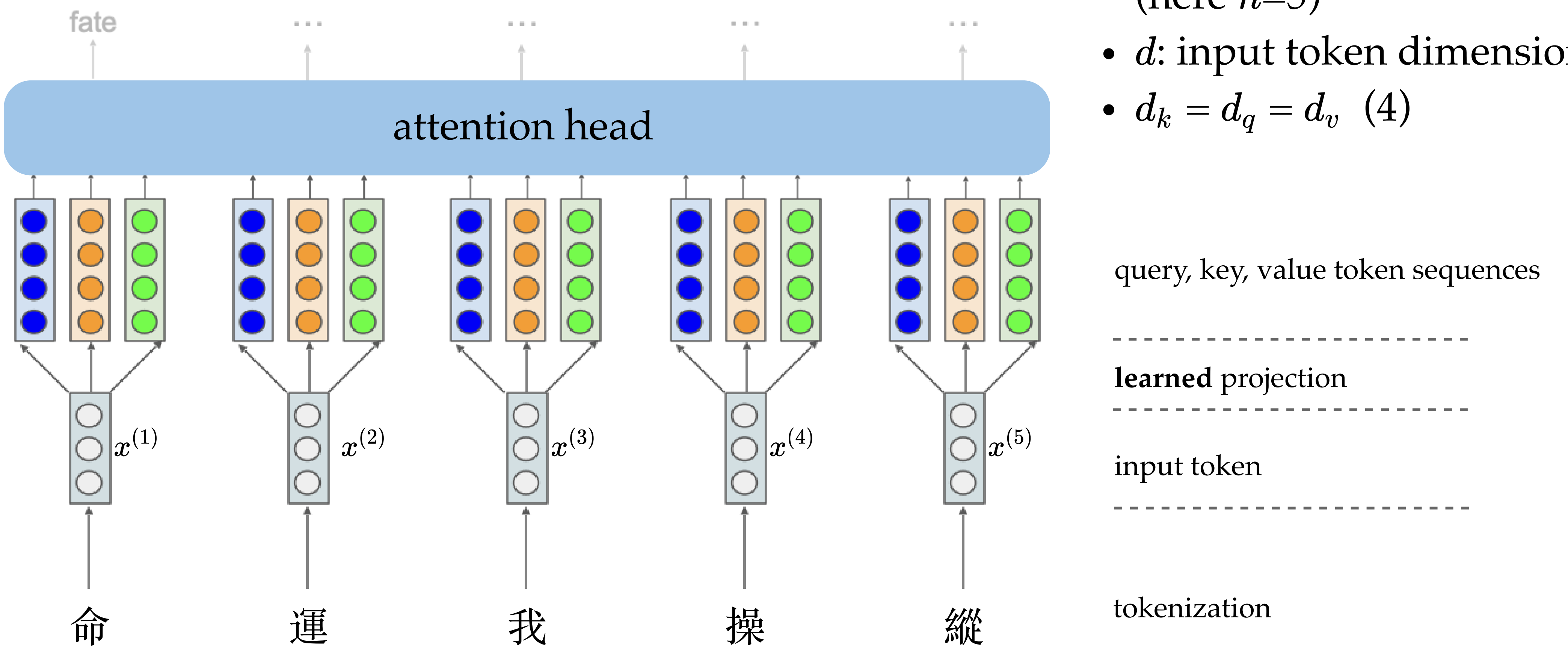


Outline

- Recap: CNN
- Transformers
  - Tokens
  - Attention
  - Self-attention
  - Learned Embedding
  - Full-stack
- (Applications and interpretation)

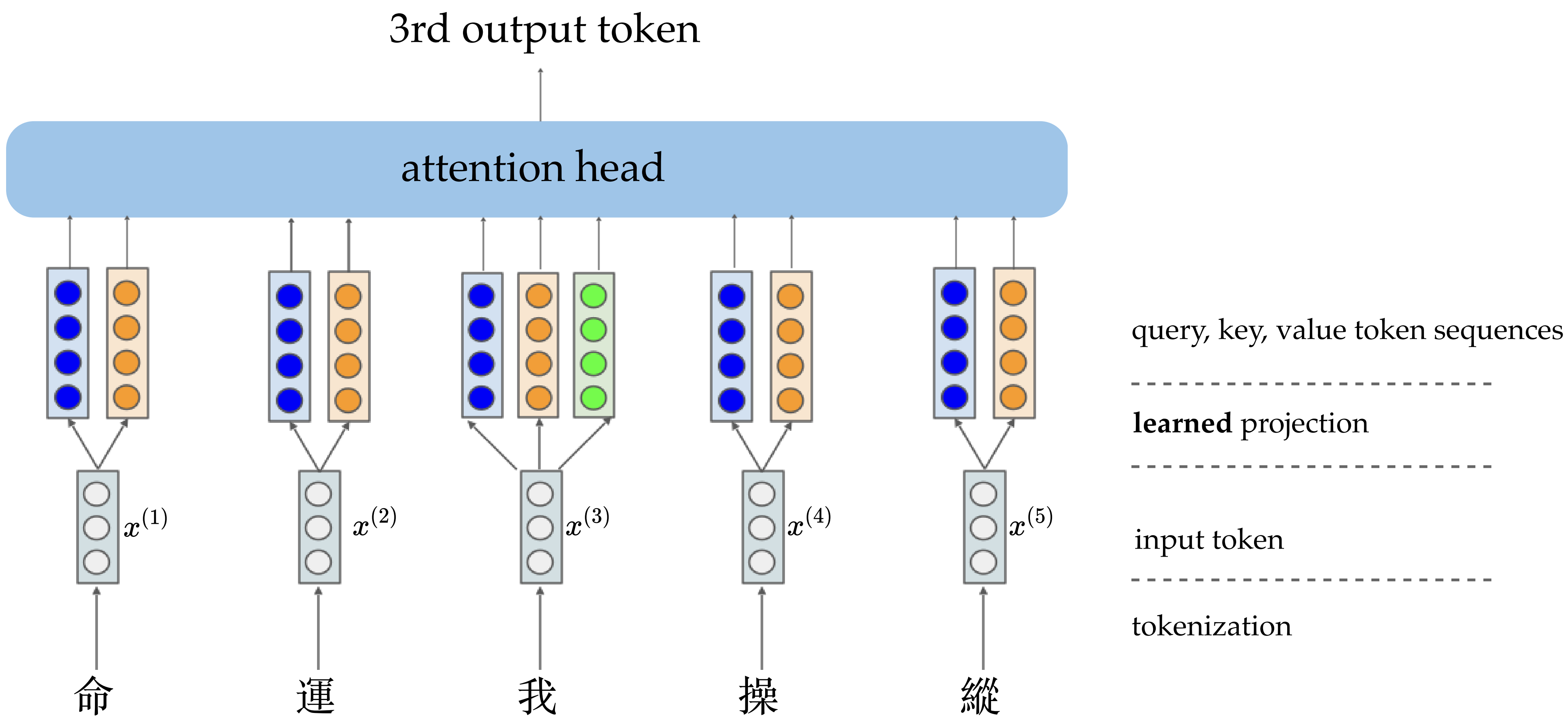
Self-attention

- query, key, value sequences: all produced by the same input sequence itself.

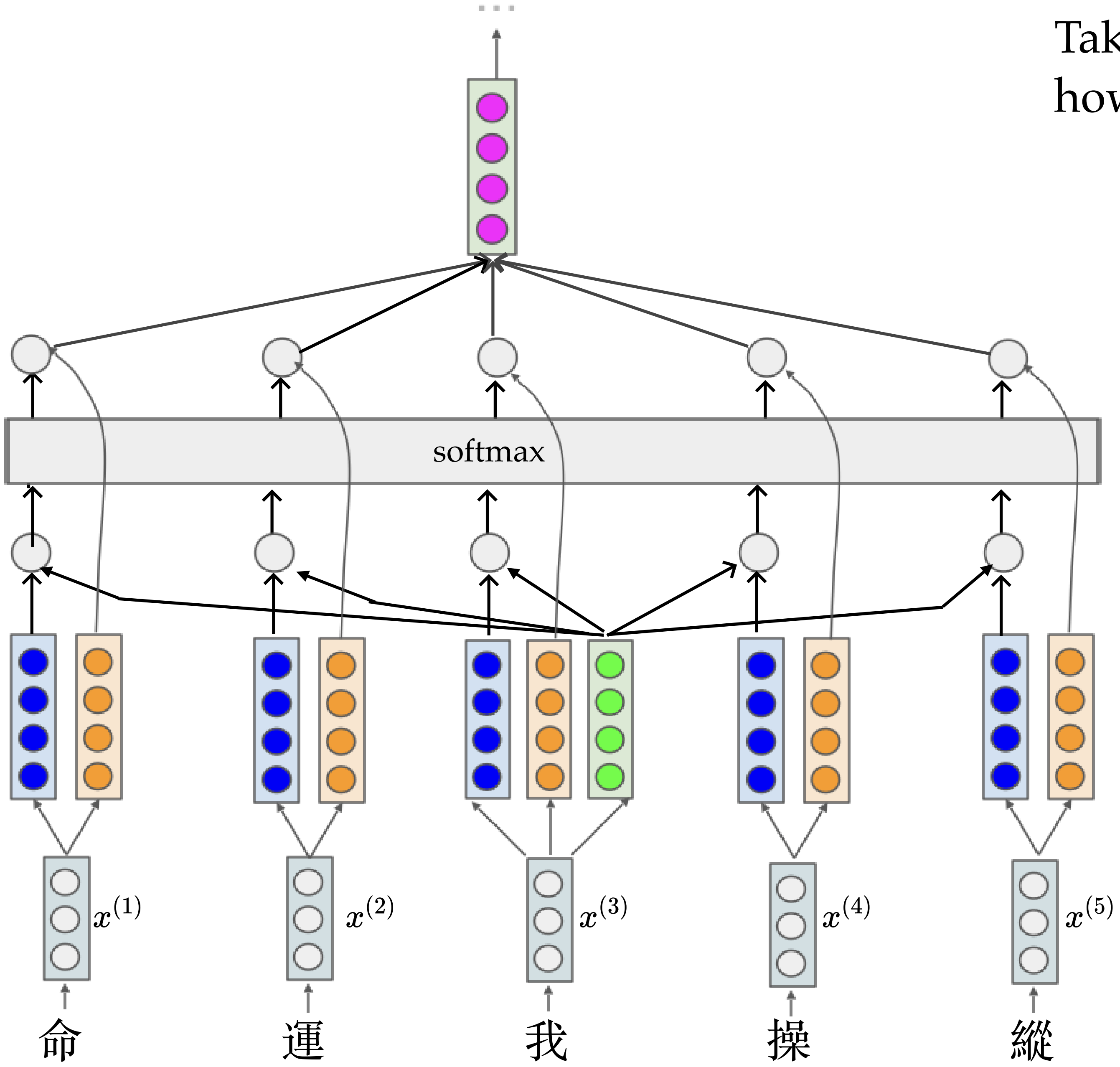


- n: number of input tokens (here n=5)
- d: input token dimension (3)
- $d_k = d_q = d_v$ (4)

- Take the 3rd input token as example, how do we get the 3rd output token?



Take the 3rd input token as example,
how do we get the 3rd output token?



the 3rd output token

one attention head

query, key, value token sequences

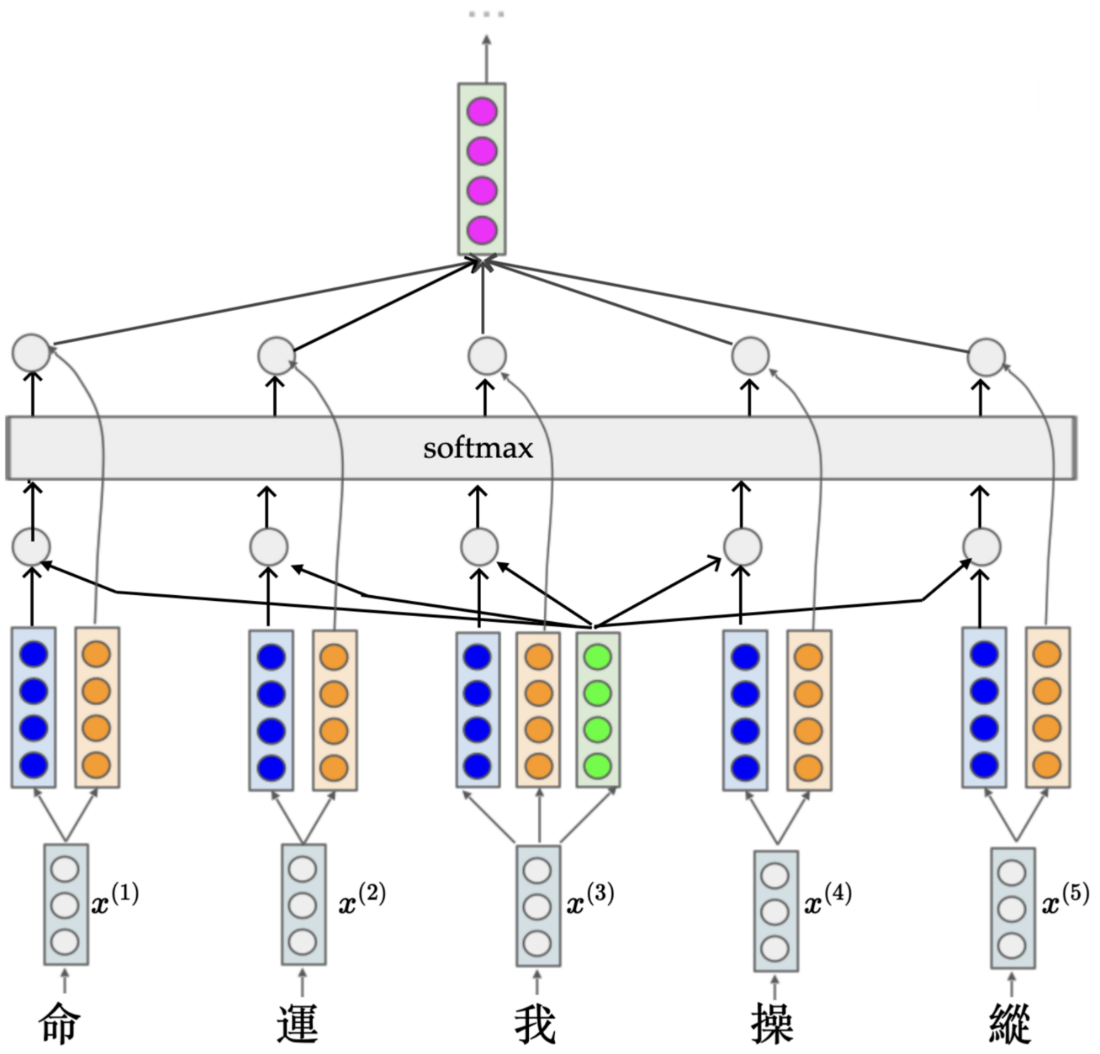
learned projection

input token

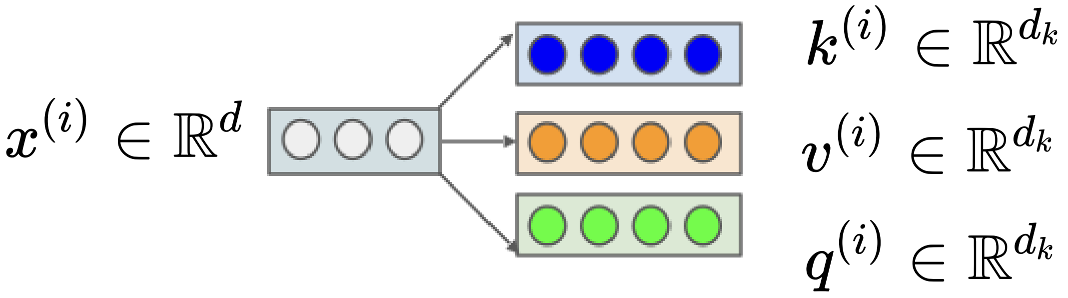
tokenization

Outline

- Recap: CNN
- Transformers
  - Tokens
  - Attention
  - Self-attention
  - Learned Embedding
  - Full-stack
- (Applications and interpretation)

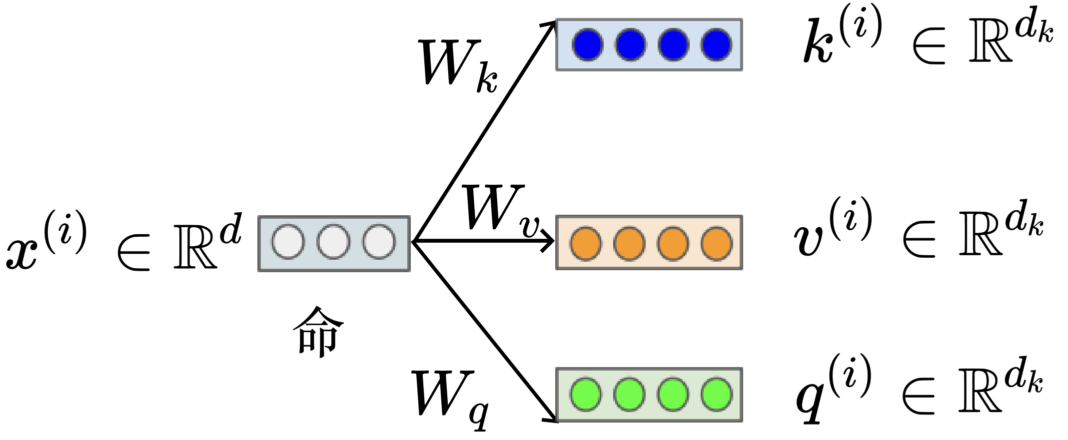


- Which color is query / key / value respectively?

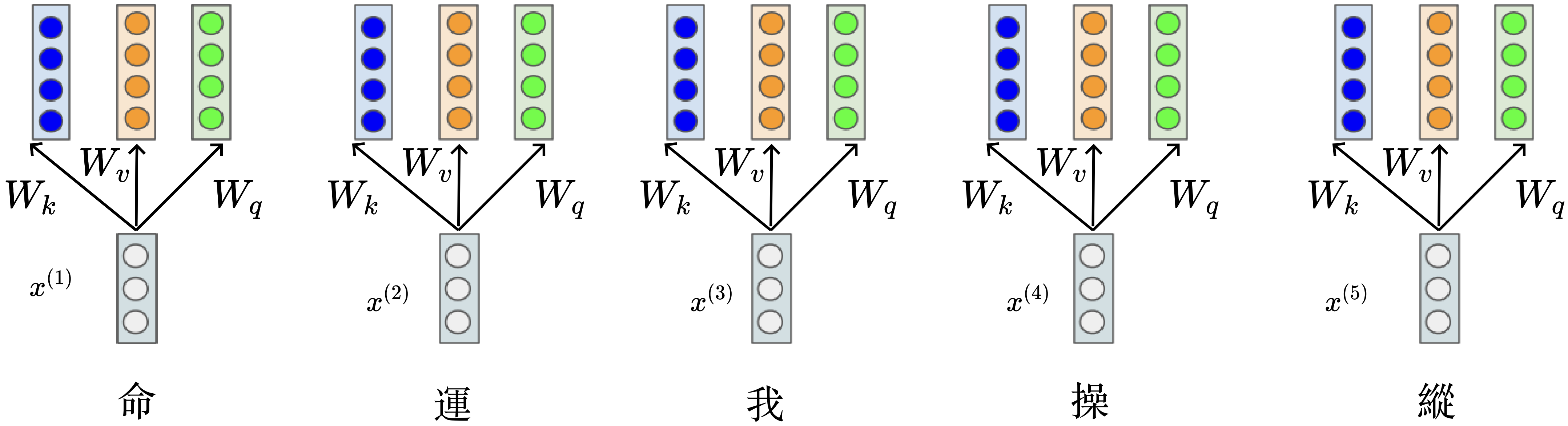


- How do we go from x to q, k, v?

  via **learned** projection weights



- Importantly, all these learned projection weights W are shared along the token sequence:

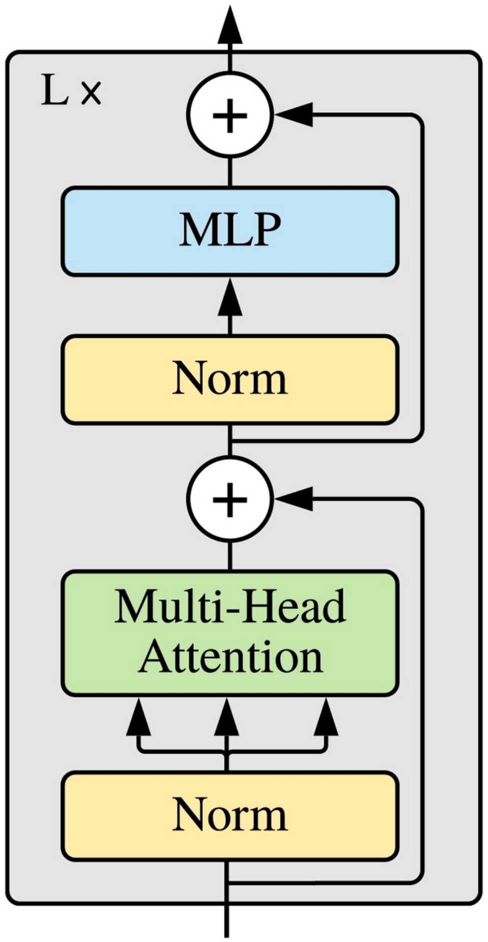


- These three weights W -- once learned -- do not change based on input token x.
- If the input sequence had been longer, we can still use the same weights in the same fashion --- just maps to a longer output sequence.
- This is yet another parallel processing (similar to convolution)
- But each (q, k, v) do depend on the corresponding input x (can be interpreted as dynamically changing convolution filter weights)

Outline

- Recap: CNN
- Transformers
  - Tokens
  - Attention
  - Self-attention
  - Learned Embedding
  - Full-stack
- (Applications and interpretation)

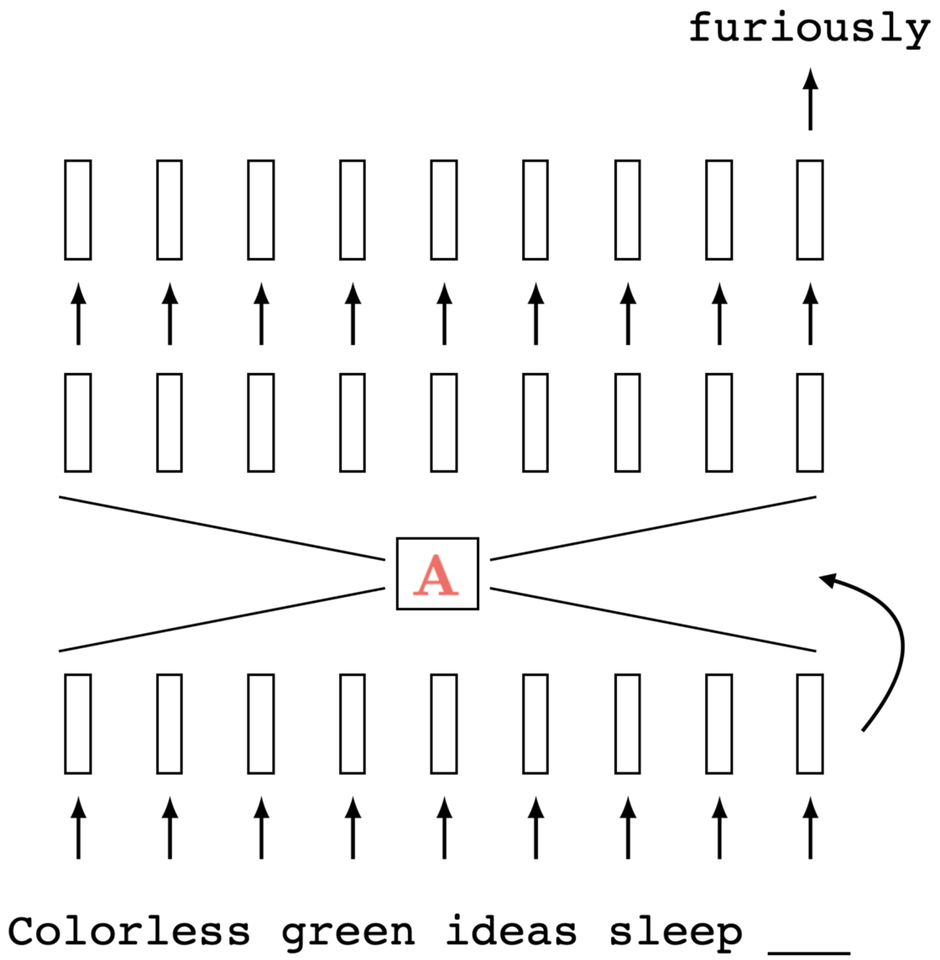
Transformers

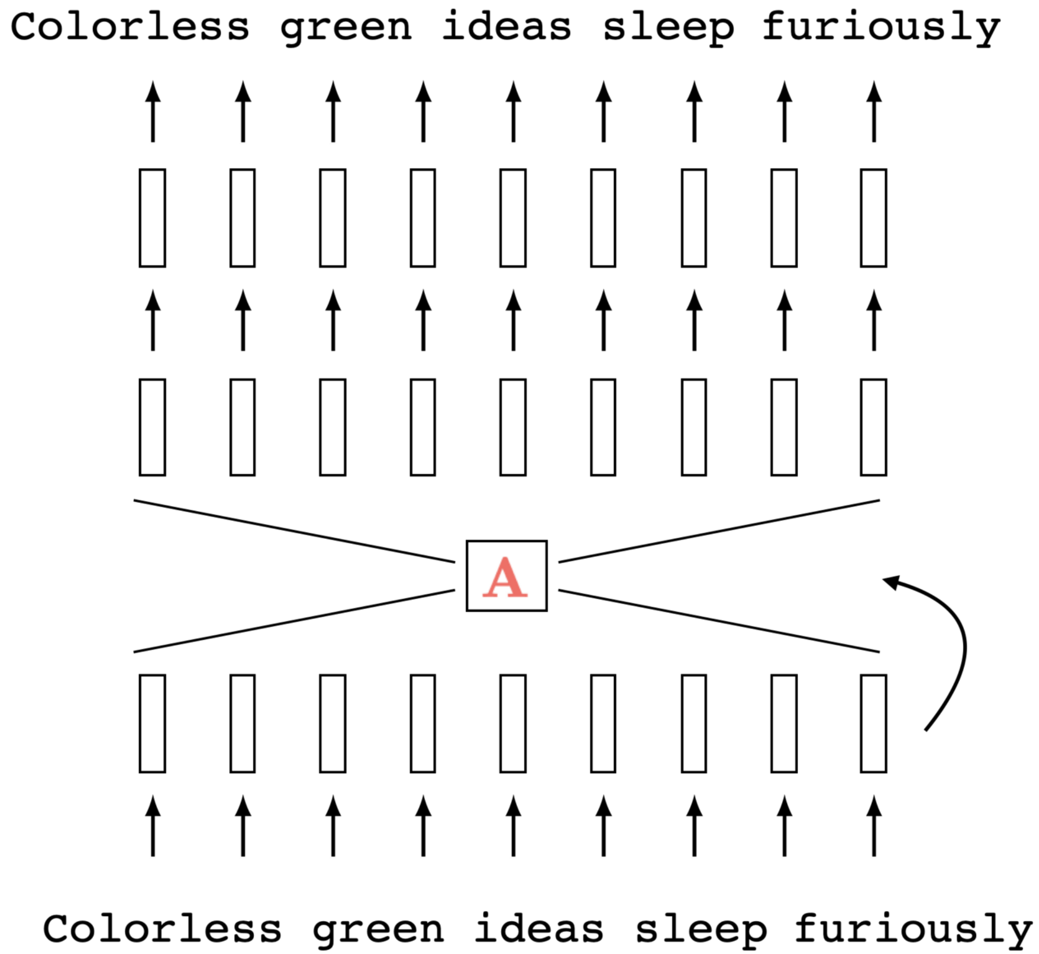


Some other ideas commonly used in practice:

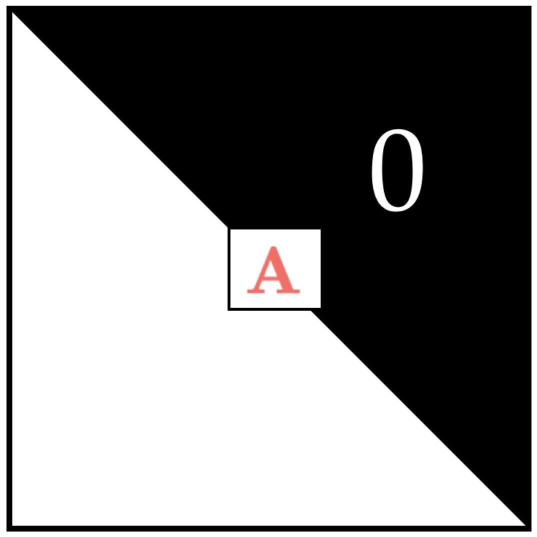
- Causal attention
- Residual connection
- Layer Normalization

Causal self-attention



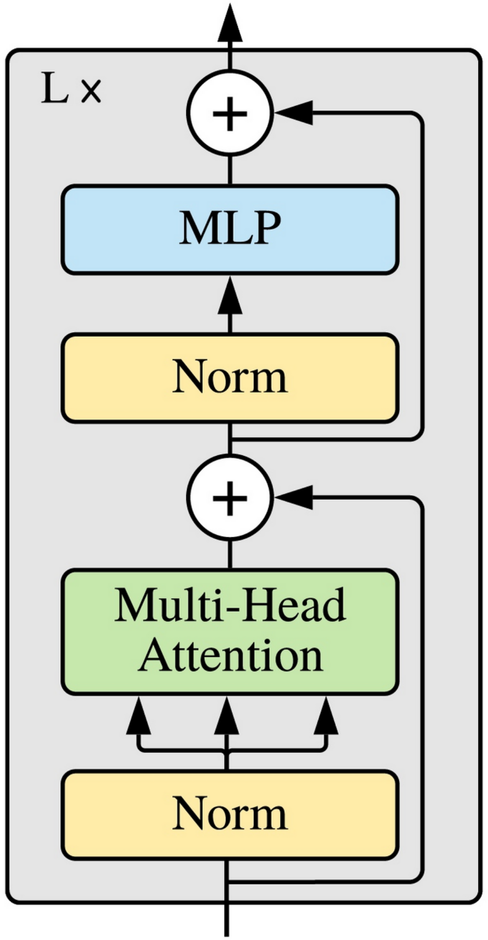


causal attention



(via masking)

Transformers



All parameters are in projection

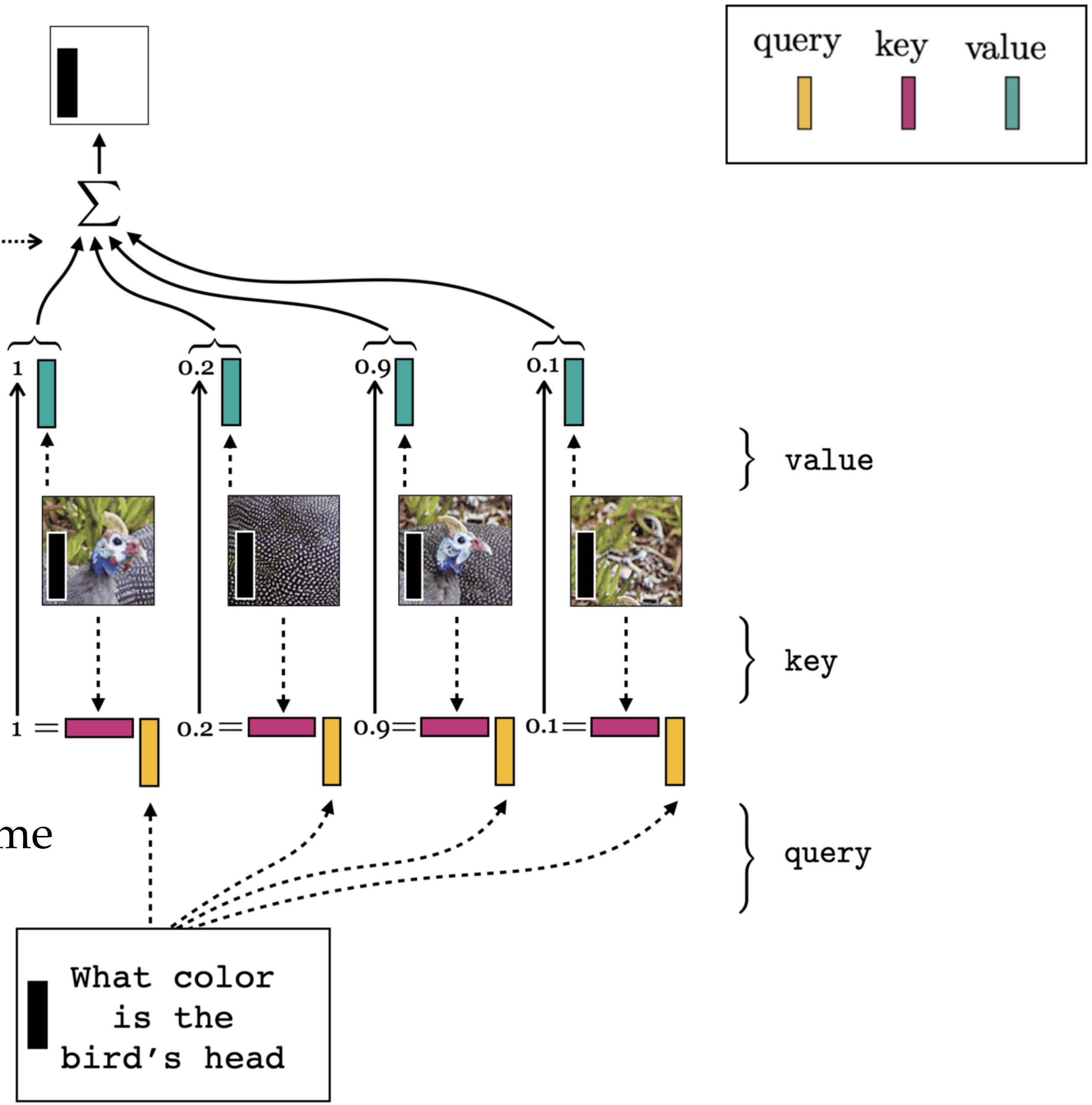
- W_q, W_k, W_v are the most specific to transforms
- MLP (i.e. fully-connected layers) could have their own weights too; same idea as week 6 NN

(

Multi-modality (text + image)

$$\mathbf{A} = \texttt{softmax}(\mathbf{s})$$

$$\mathbf{s} = [\mathbf{q}_{\texttt{question}}^T \mathbf{k}_1, \ldots, \mathbf{q}_{\texttt{question}}^T \mathbf{k}_N]$$



- notice how query and (keyvalue) come from different inputs?
- similar to the dictionary example
- unlike the self-attention translation example.

Success mode:

Failure mode:

Giannis Daras ✈️ NeurIPS 2023
@giannis_daras

DALLE-2 has a secret language.
"Apoploe vesrreaitais" means birds.
"Contarra ccetnxniams luryca tanniounons" means bugs or pests.

The prompt: "Apoploe vesrreaitais eating Contarra ccetnxniams luryca tanniounons" gives images of birds eating bugs.

A thread (1/n)🧵

Another example: "Two whales talking about food, with subtitles". We get an image with the text "Wa ch zod rea" written on it. Apparently, the whales are actually talking about their food in the DALLE-2 language. (4/n)

Figure 4: Left: Image generated with the prompt: "`Two whales talking about food, with subtitles.`". Right: Images generated with the prompt: "`Wa ch zod ahaakes rea.`". The gibberish language, "`Wa ch zod ahaakes rea.`", produces images that are related to the text-conditioning and the visual output of the first image.

)

We'd love it for you to share some lecture feedback.

Thanks
(for your attention :)!