Self Attention

NLP: Fall 2023

Anoop Sarkar

Preliminaries

LayerNorm

https://arxiv.org/abs/1607.06450

also see: https://arxiv.org/abs/1911.07013

$$\mathbf{x} = (x_1, x_2, \ldots, x_H)$$

$$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i \qquad \sigma^2 = \frac{1}{H}\sum_{i=1}^{H} (x_i - \mu)^2$$

$$N(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sigma + \epsilon}$$

ϵ avoids div by zero

$$\mathbf{h} = \mathbf{g} \cdot N(\mathbf{x}) + \mathbf{b}$$

g and **b** are hyperparameters with dimension H

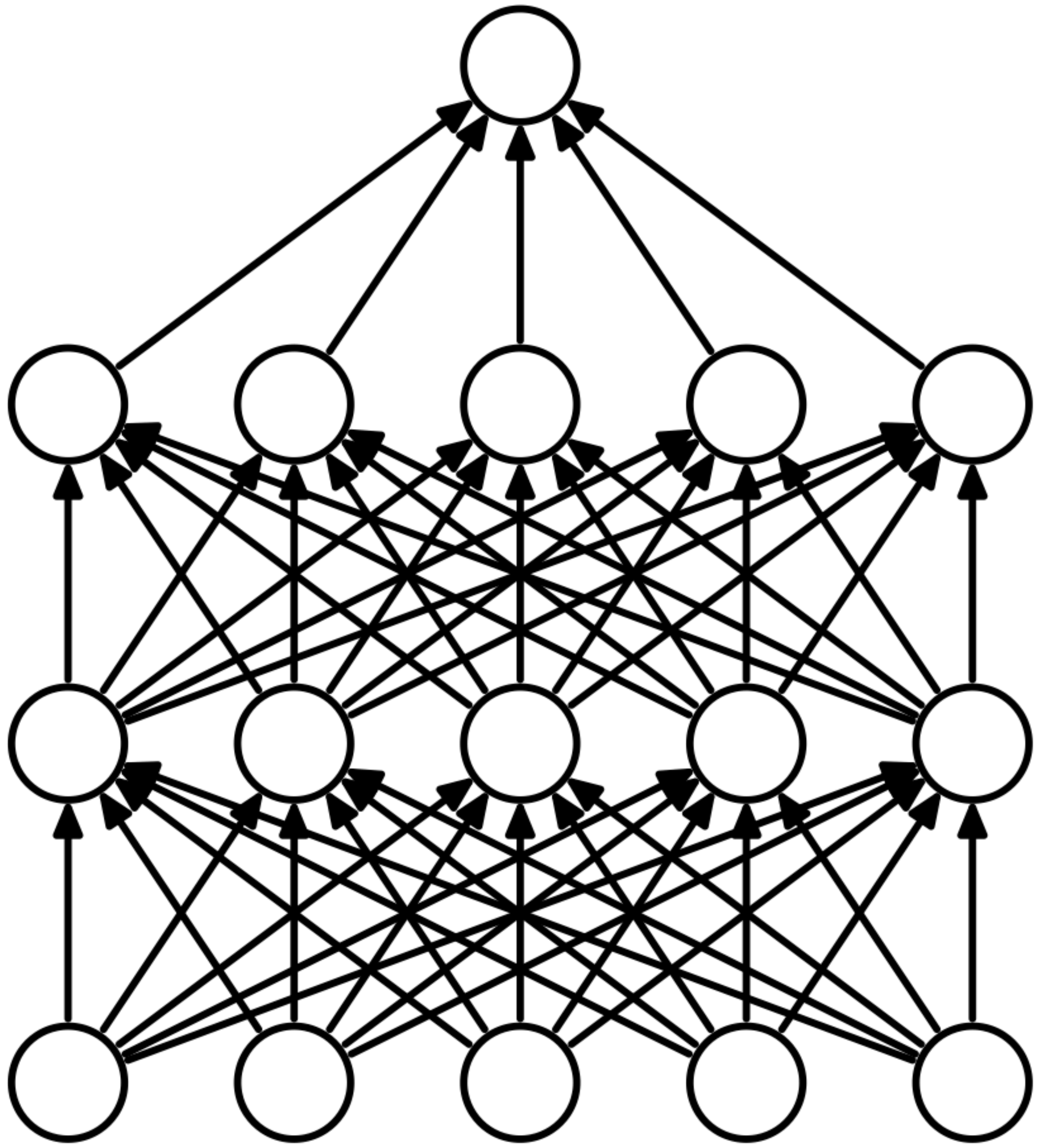
In PyTorch

```
>>> # NLP Example
>>> batch, sentence_length, embedding_dim = 20, 5, 10
>>> embedding = torch.randn(batch, sentence_length, embedding_dim)
>>> layer_norm = nn.LayerNorm(embedding_dim)
>>> # Activate module
>>> layer_norm(embedding)
```

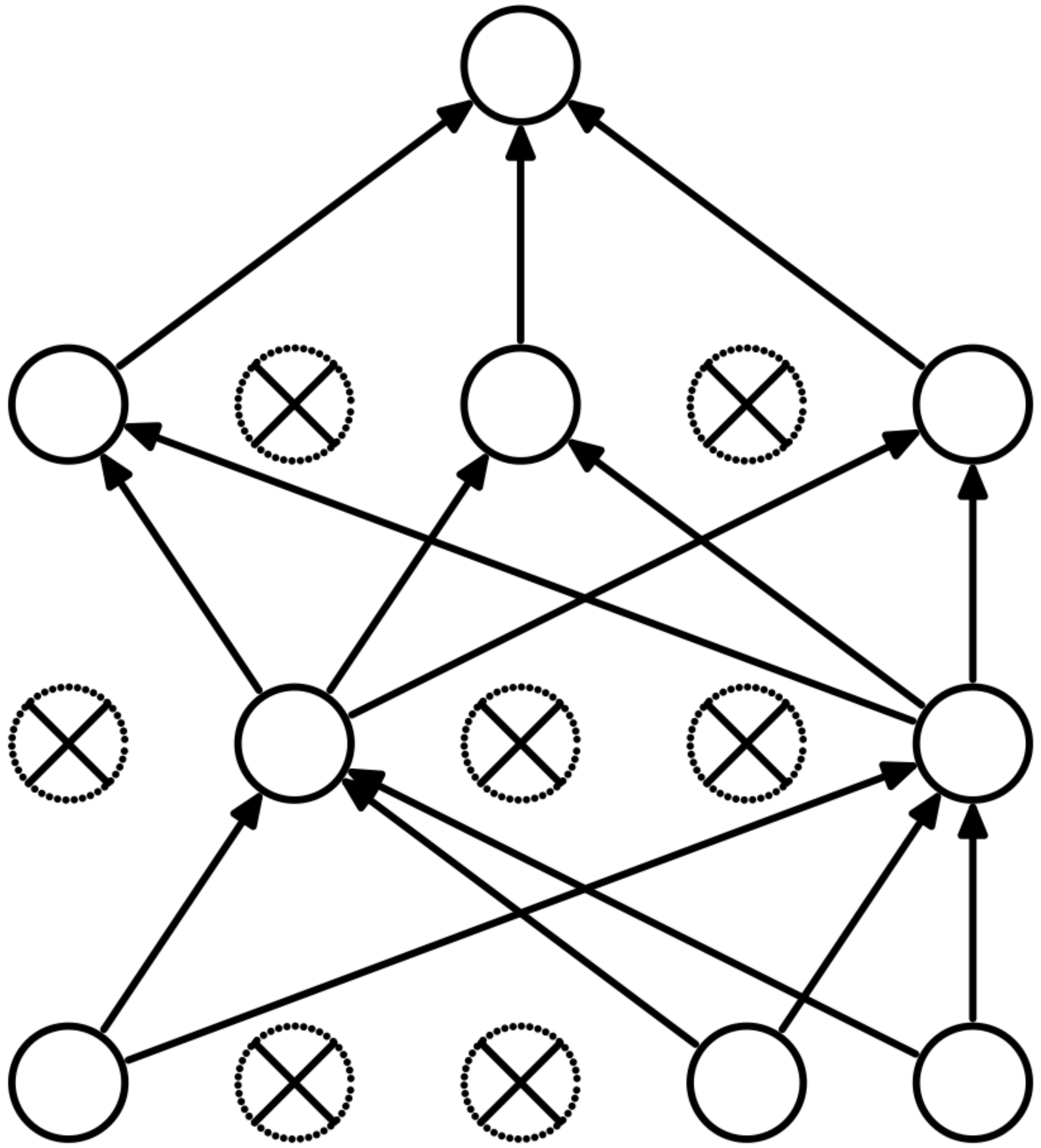
Dropout

https://jmlr.org/papers/v15/srivastava14a.html
https://arxiv.org/abs/1207.0580

aka how to train 2^n neural networks when it has n units

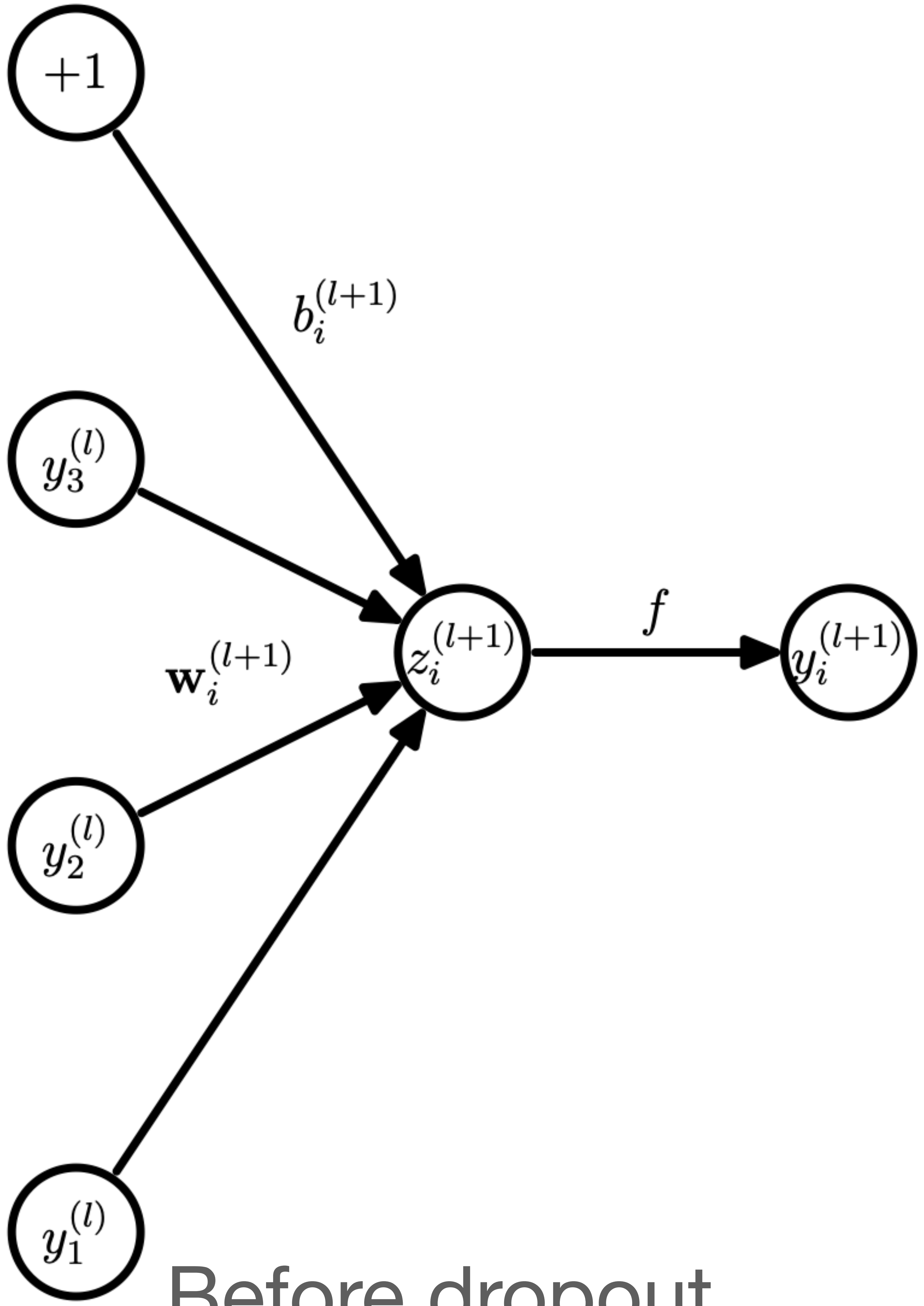


(a) Standard Neural Net

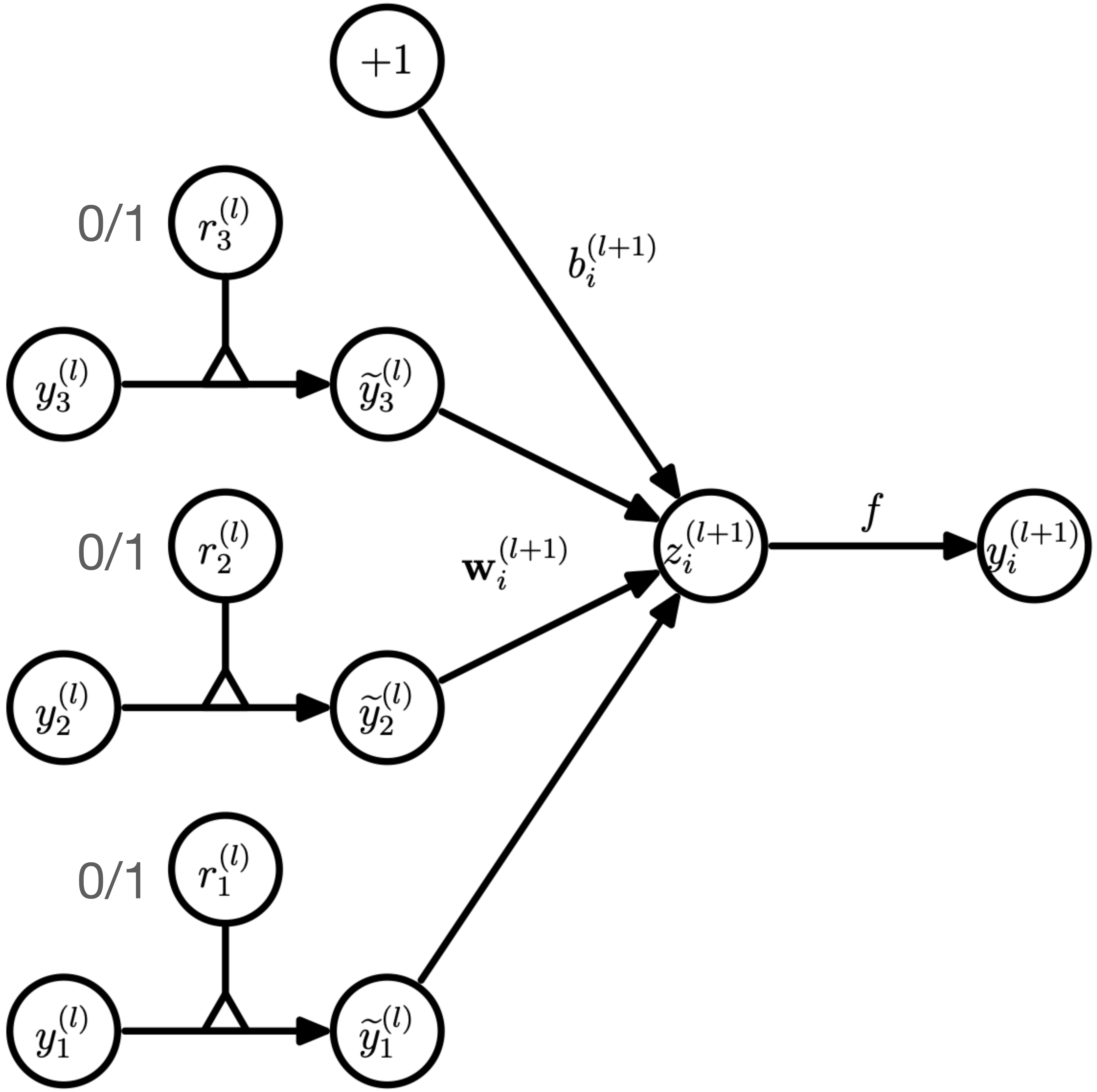


(b) After applying dropout.

- Dropout helps to avoid over-fitting to training
- It cannot rely on a small set of simple "features" to make accurate predictions.
- Similar to "regularization" which usually uses constraints over weight values.



Before dropout



After dropout

Before dropout

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)},$$
$$y_i^{(l+1)} = f(z_i^{(l+1)}),$$

After dropout

$$r_j^{(l)} \sim \text{Bernoulli}(p), \quad 0/1$$
$$\widetilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)},$$
$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^l + b_i^{(l+1)},$$
$$y_i^{(l+1)} = f(z_i^{(l+1)}).$$

In PyTorch

```
>>> m = nn.Dropout(p=0.2)
>>> input = torch.randn(20, 16)
>>> output = m(input)
```

default: 0.5

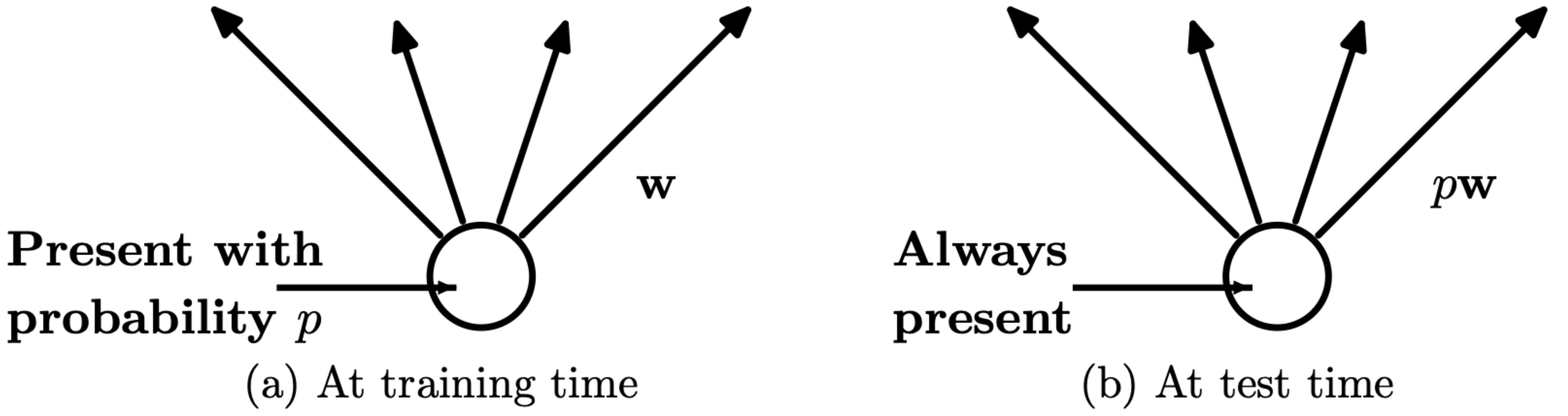


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

In Pytorch the outputs are scaled by a factor of $\frac{1}{1-p}$ during training so at inference/test/evaluation time the dropout function simply computes the identity function

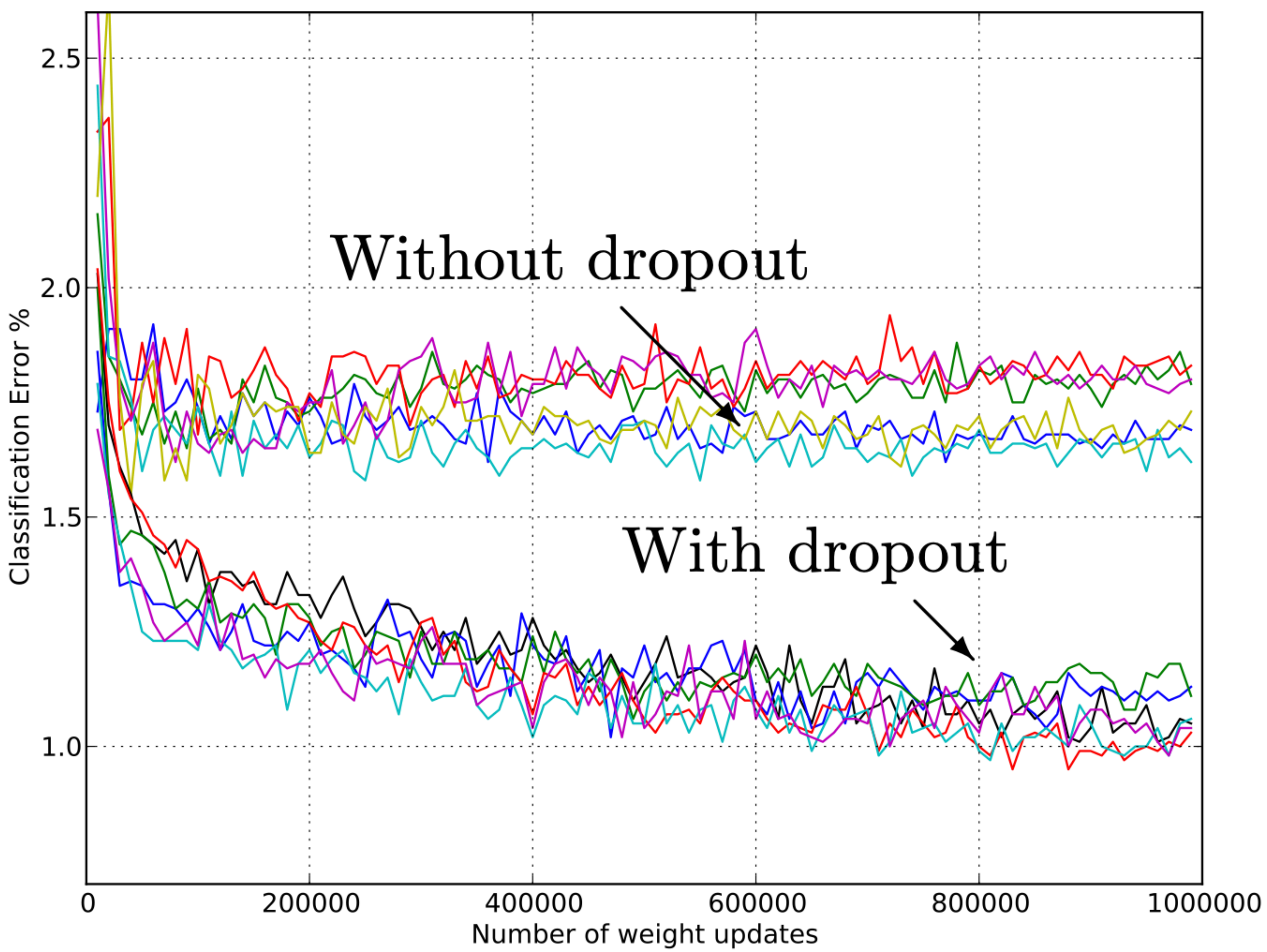
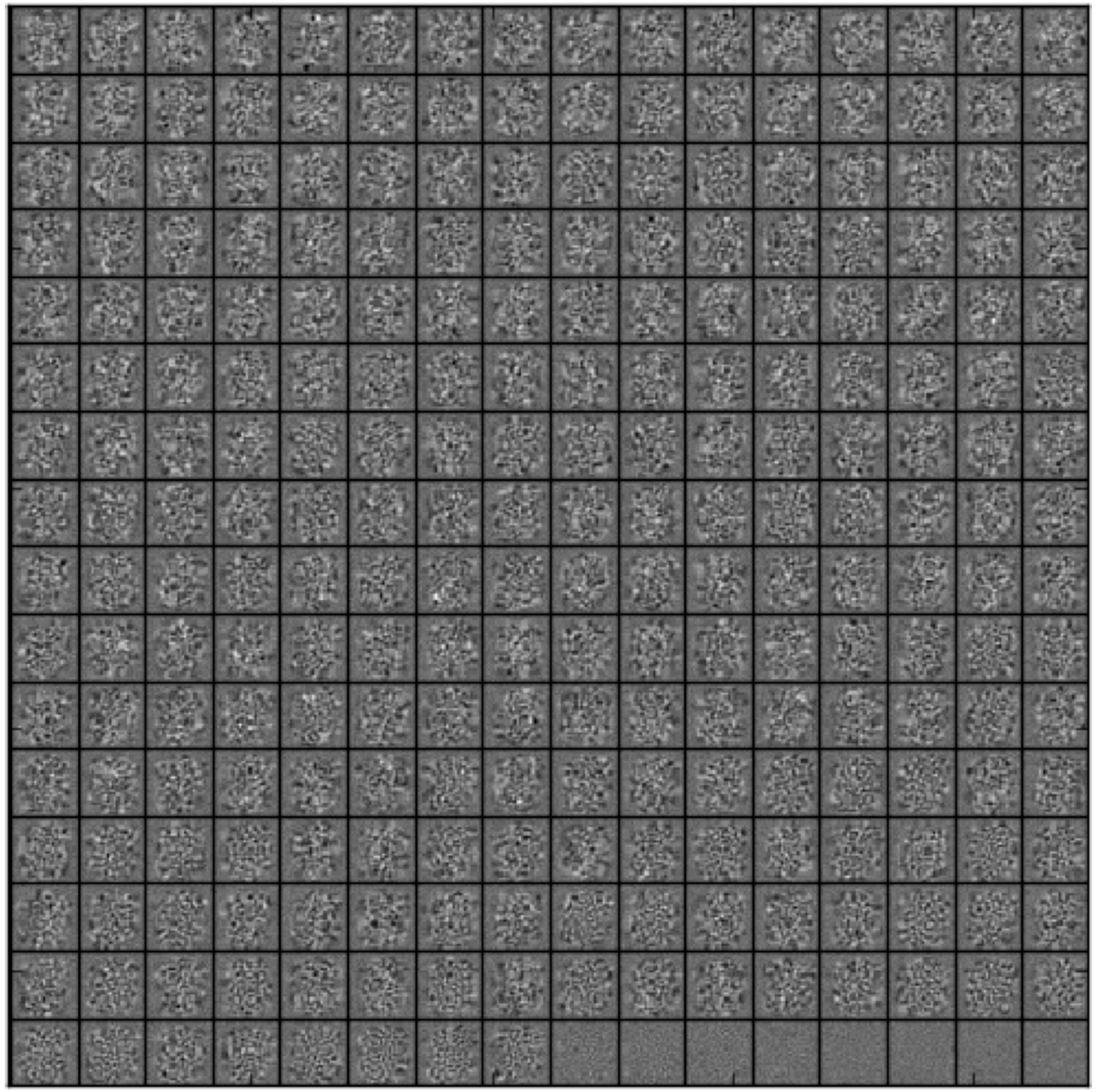
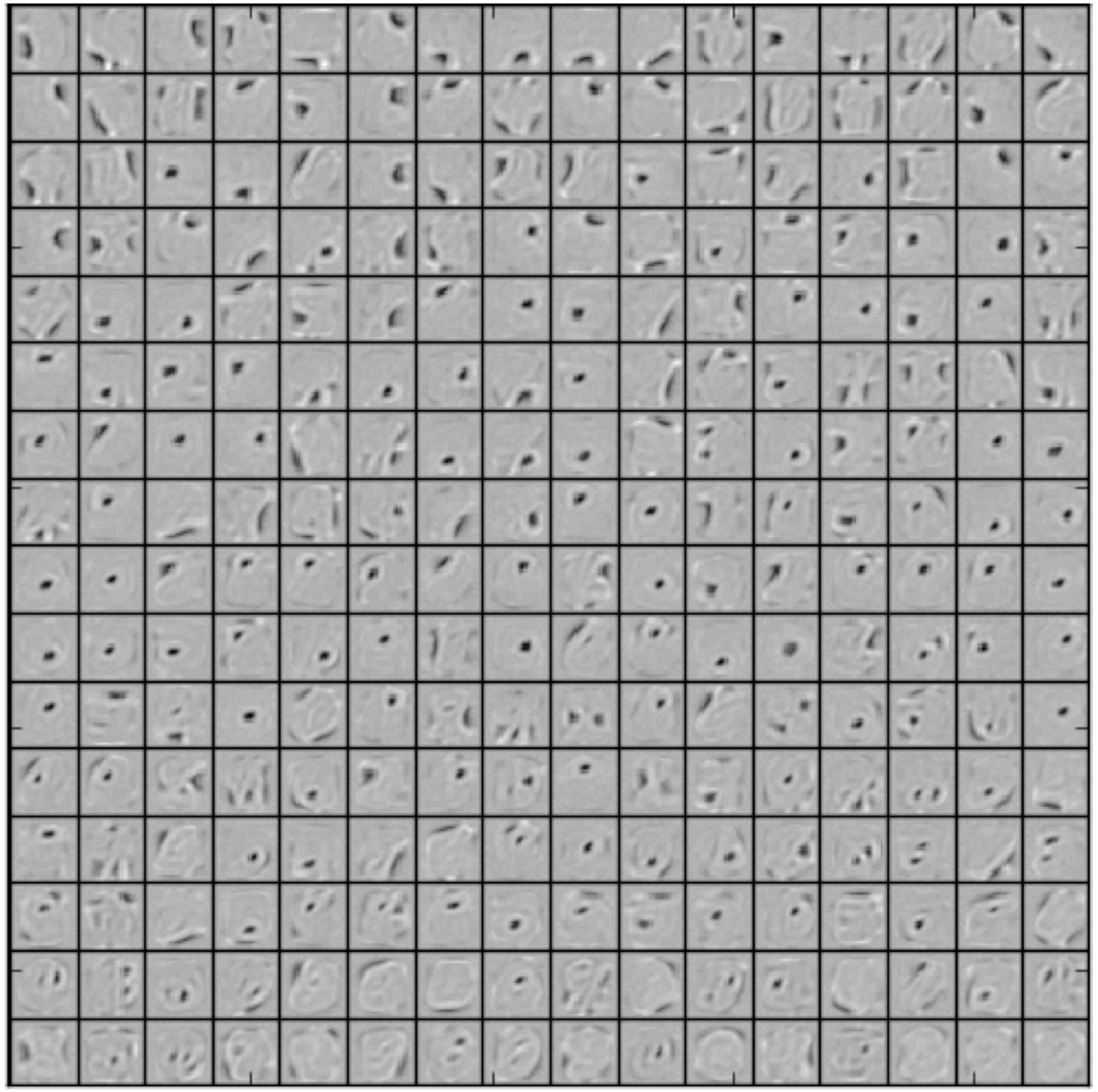


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

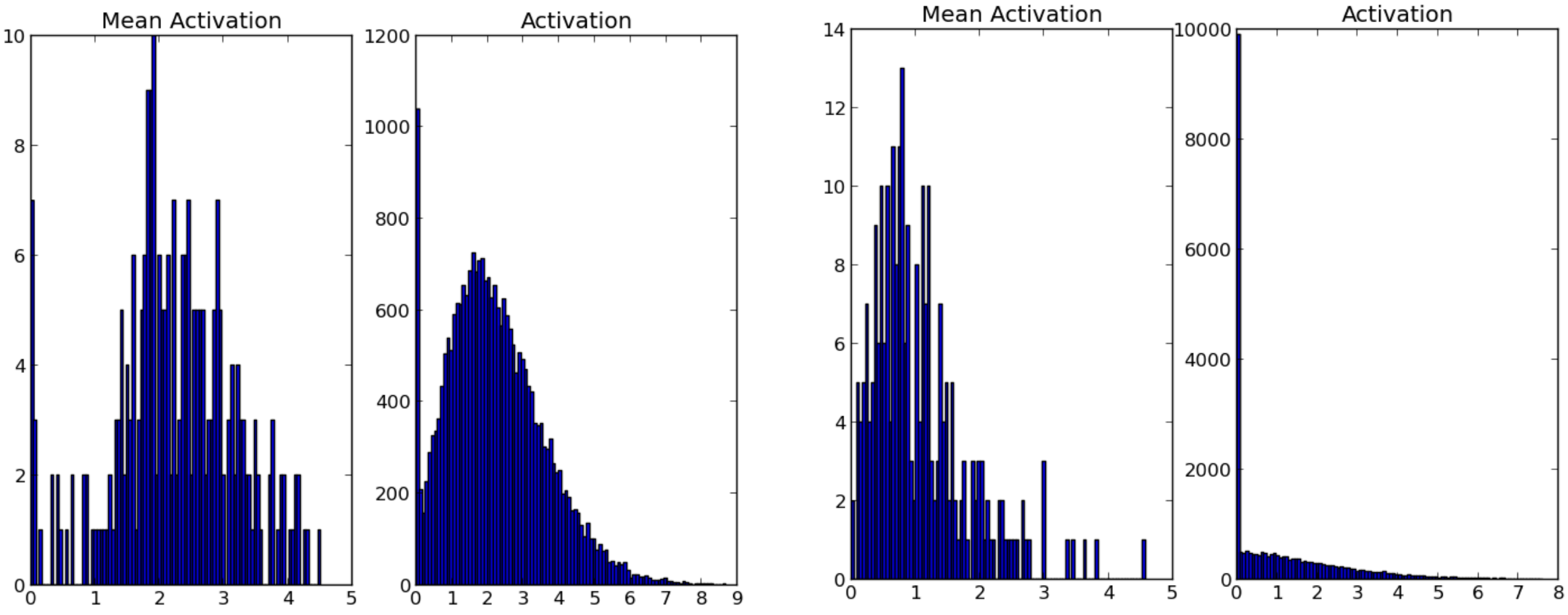


(a) Without dropout



(b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.



(a) Without dropout (b) Dropout with $p = 0.5$.

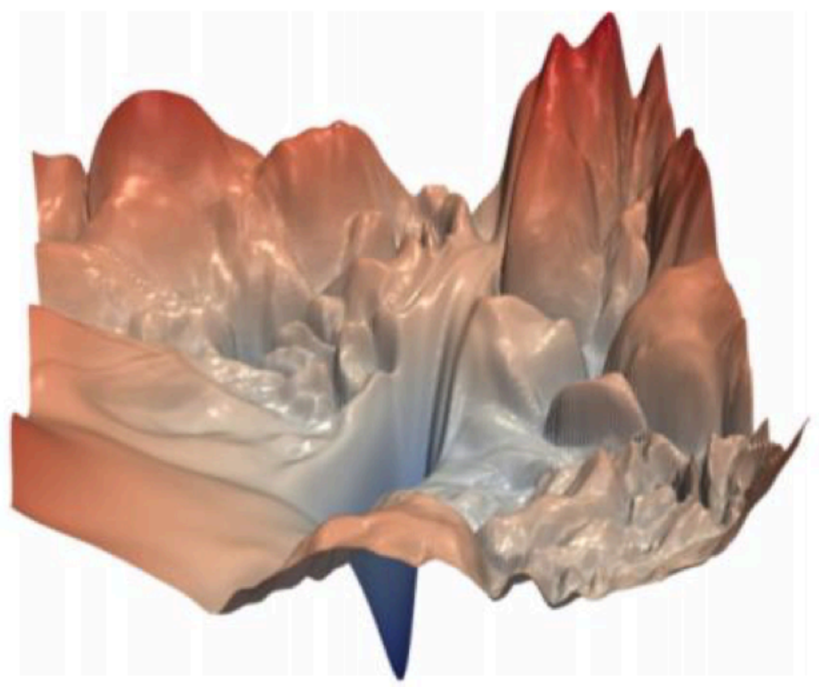
Figure 8: Effect of dropout on sparsity. ReLUs were used for both models. **Left**: The histogram of mean activations shows that most units have a mean activation of about 2.0. The histogram of activations shows a huge mode away from zero. Clearly, a large fraction of units have high activation. **Right**: The histogram of mean activations shows that most units have a smaller mean mean activation of about 0.7. The histogram of activations shows a sharp peak at zero. Very few units have high activation.

He+ 2015 https://arxiv.org/abs/1512.03385

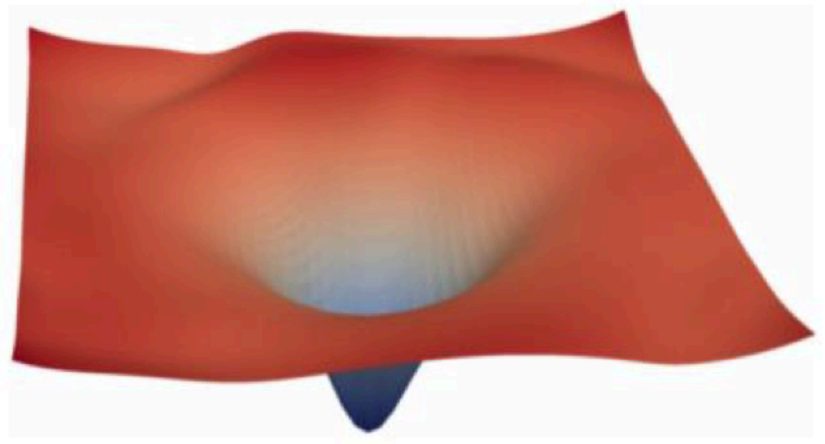
Residual Connections

Add input of a layer to output of that layer

- $\mathbf{z}^{\ell+1} = f(\mathbf{z}^{\ell}) + \mathbf{z}^{\ell}$
- Local gradient is 1 for the identity function
- Easier to learn the difference from the identity function than to learn the function from scratch.



[no residuals]



[residuals]

https://arxiv.org/pdf/1712.09913.pdf

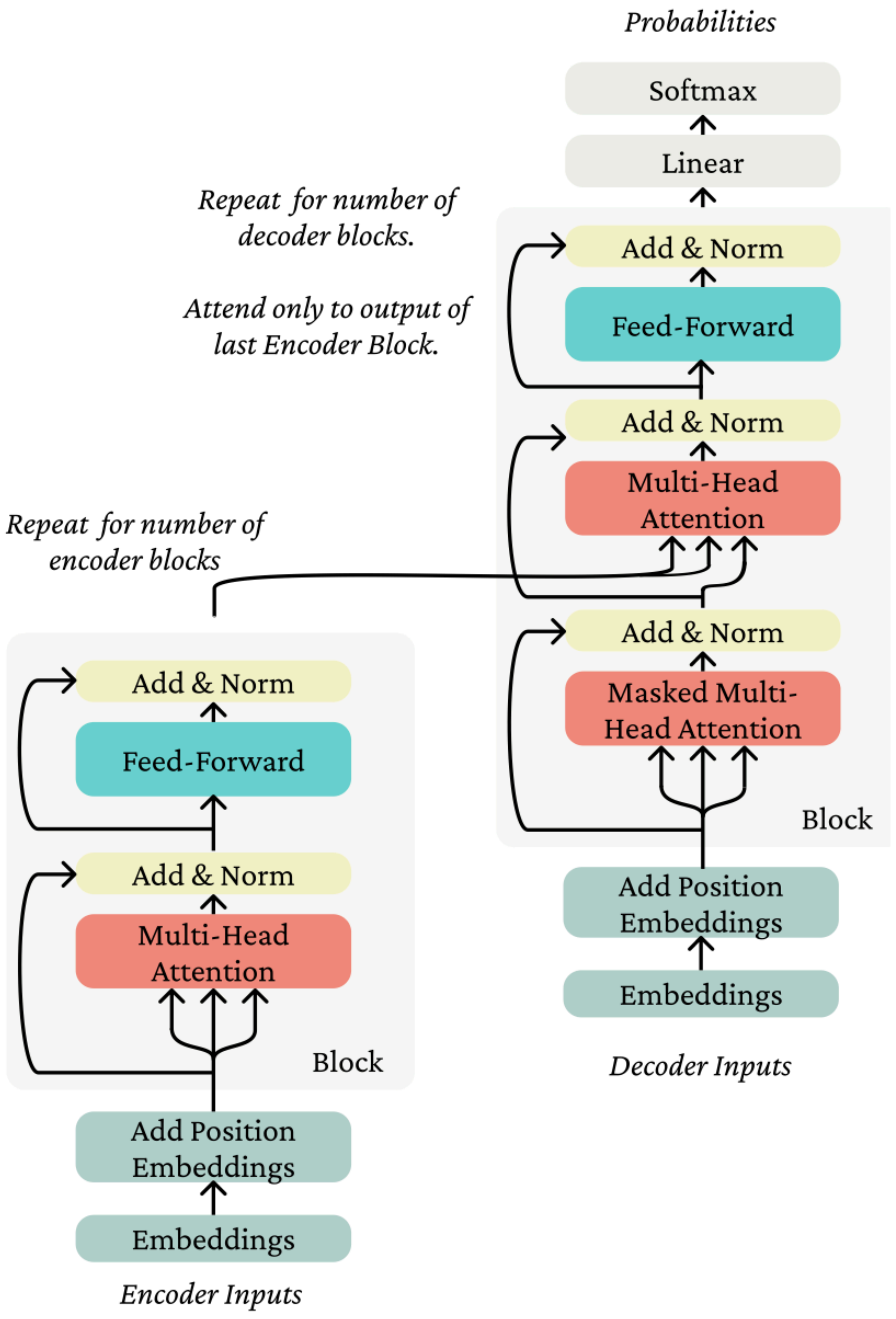
Transformer Encoder-Decoder

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

https://arxiv.org/abs/1409.0473

NIPS (2017)



Transformer Encoder-Decoder

cs224n-self-attention-transformers-2023_draft.pdf

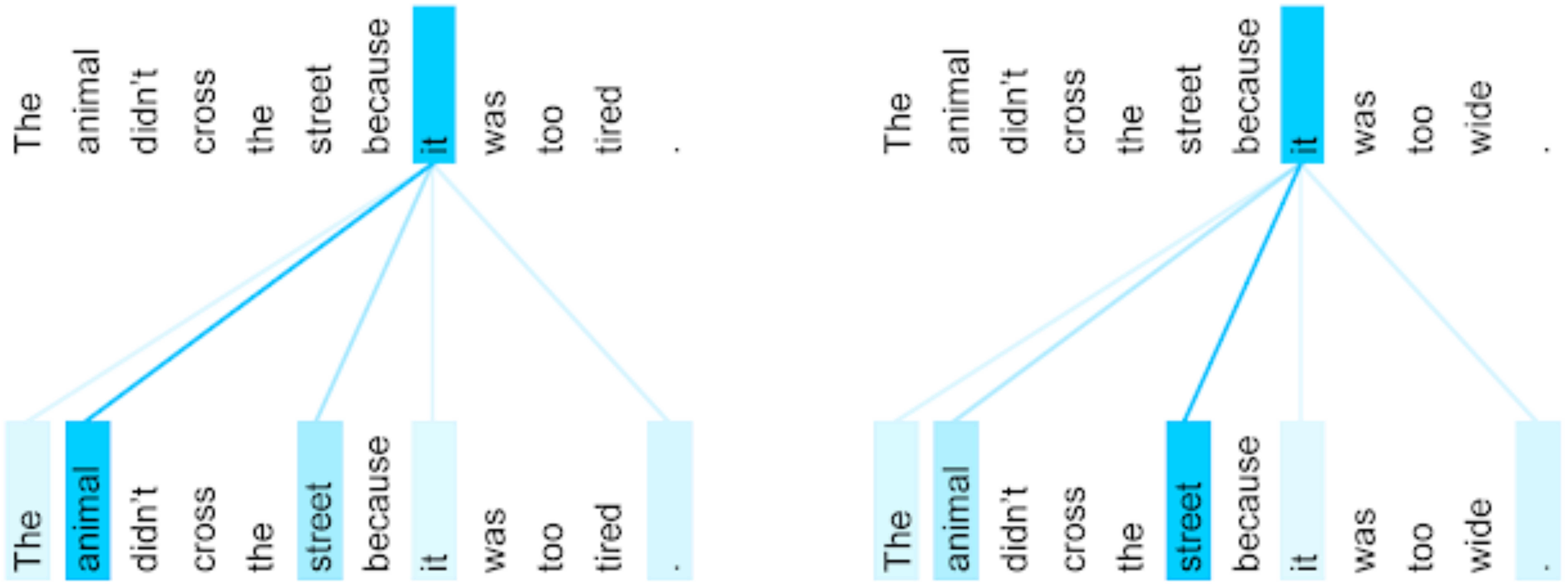
https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.

The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.

the translation for “it” depends on the gender of the noun it refers to - and in French “animal” and “street” have different genders

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html



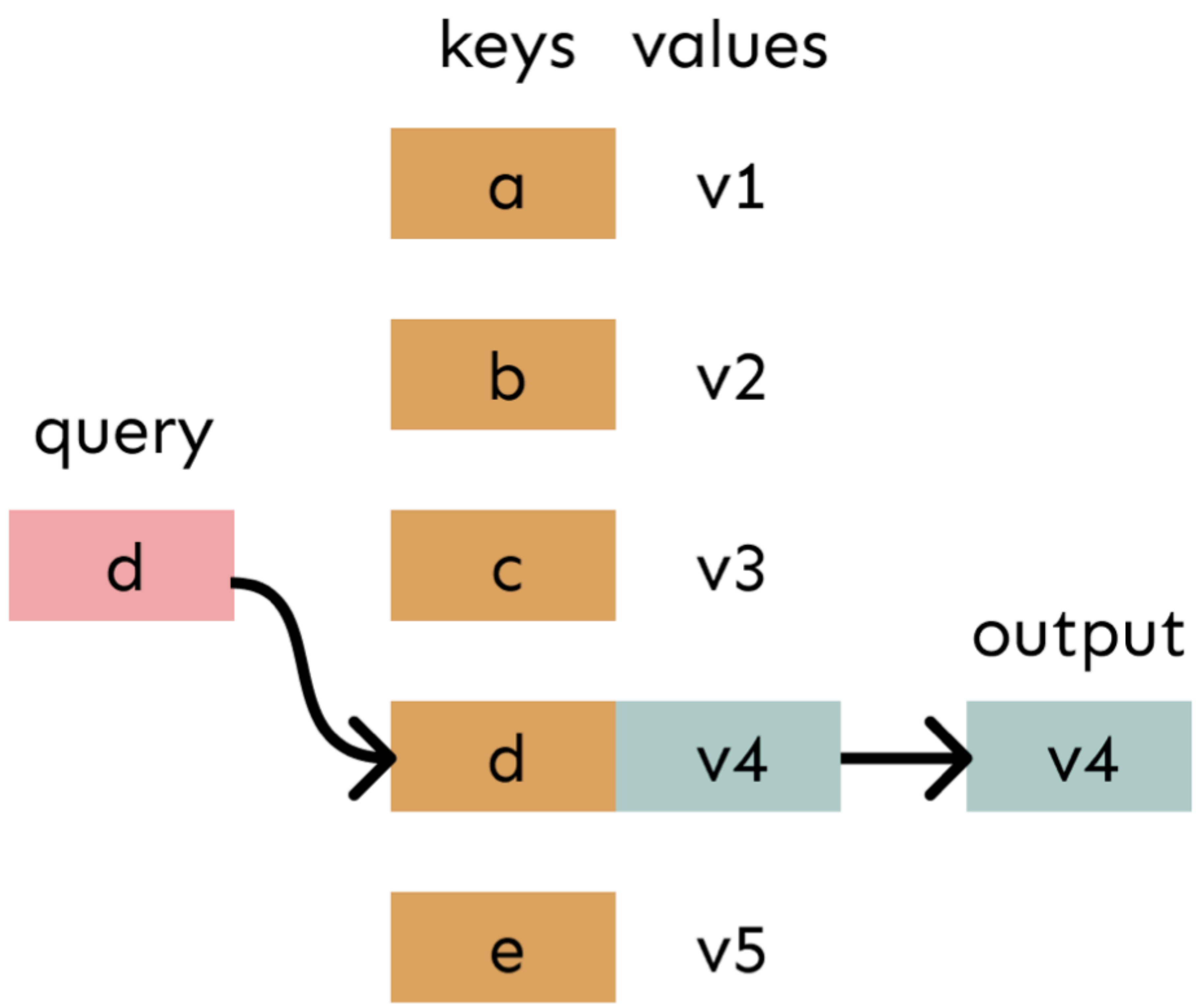
The encoder self-attention distribution for the word “it” from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

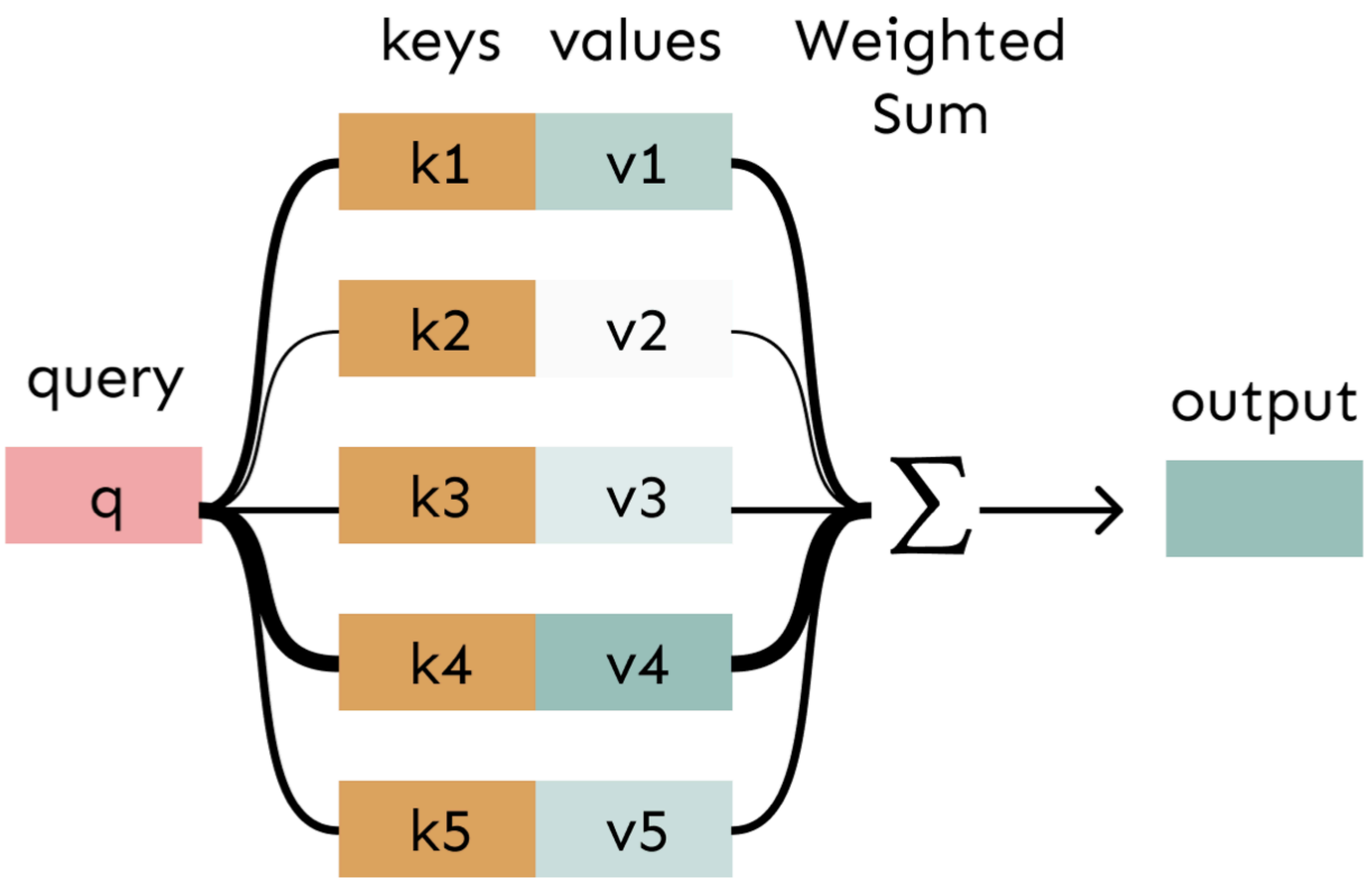
Self Attention

- Take a query vector (based on one token)
- Do a "**soft lookup**" in a key-value store; **pick up** the key **most like** the query and return the value vector
- "**pick up**" = return the average value based on a probability distribution
- "**most like**" = higher probability for a key means it is **more like** the query
- "**more like**" = dot product e.g.
- In *self attention* we use the same tokens for queries, keys and values

cs224n-self-attention-
transformers-2023_draft.pdf



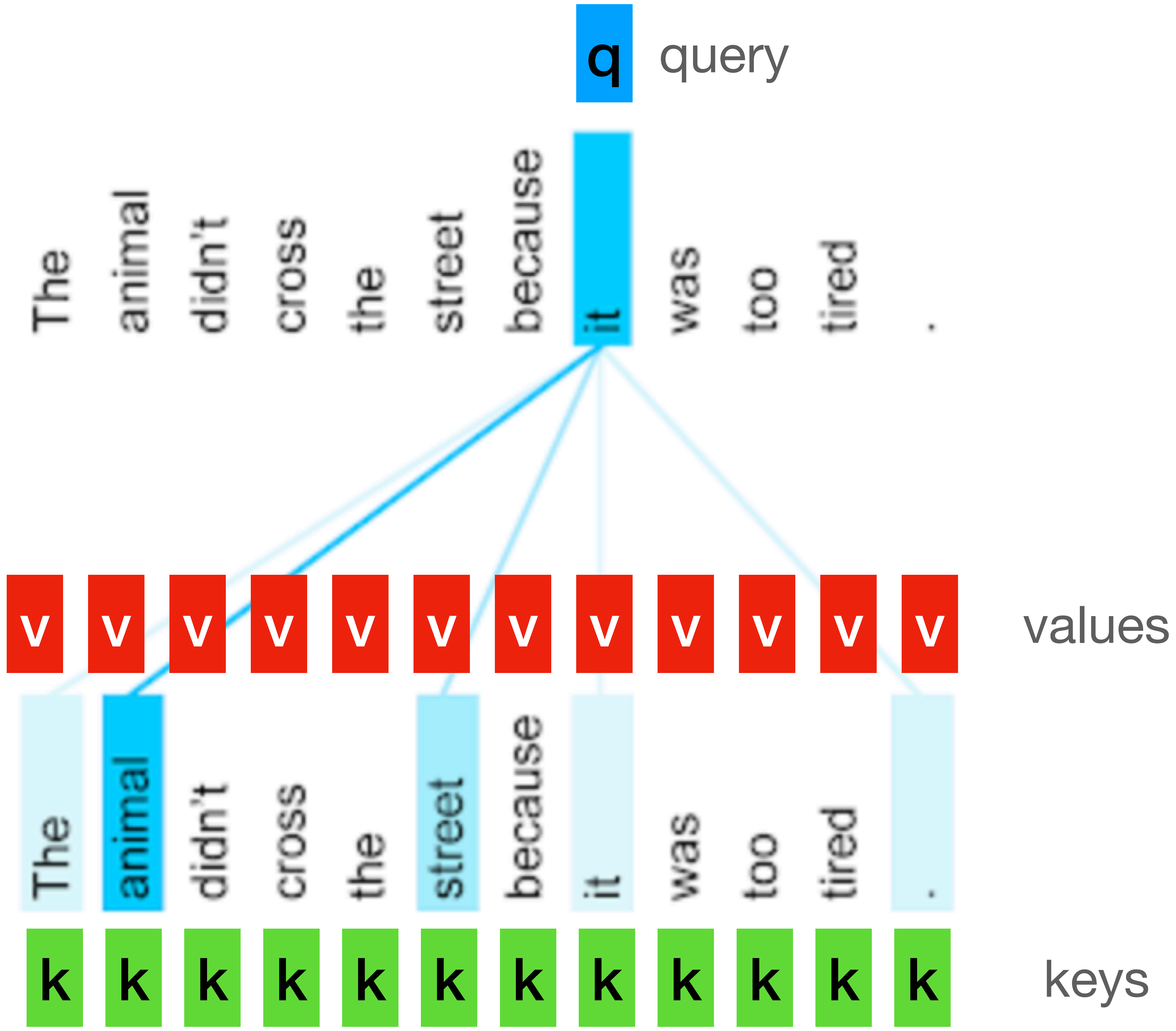
Standard key-value lookup



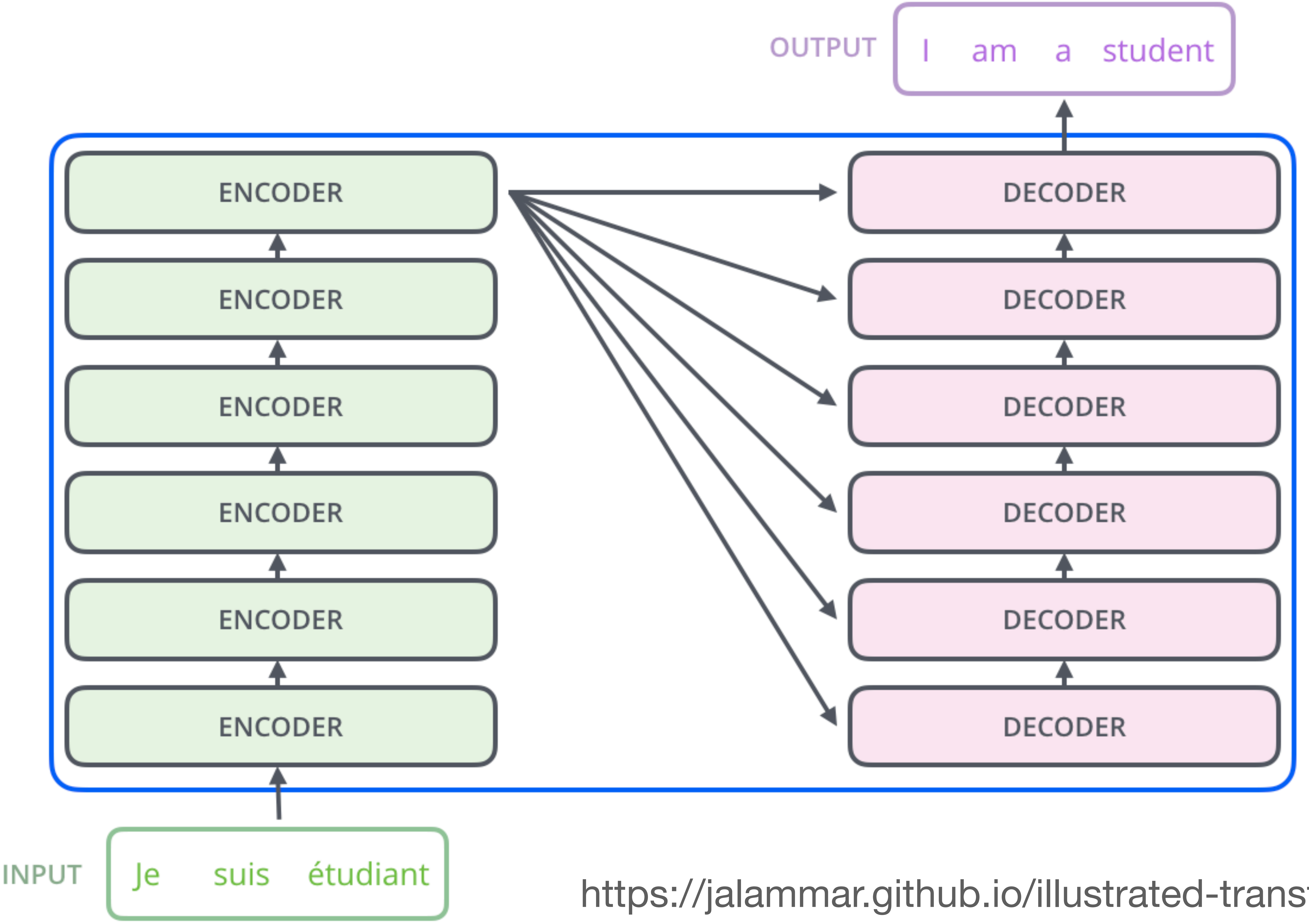
Self attention key-value lookup

cs224n-self-attention-
transformers-2023_draft.pdf

Self Attention



https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html



https://jalammar.github.io/illustrated-transformer/

Self-attention

keys, queries and values from the same sequence

- Let $\mathbf{w} = (w_1, \ldots, w_n)$ be a sequence of tokens, like "Havana is the capital of"
- For each w_i let $\mathbf{x}_i = E\mathbf{w}_i$ where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix. V is the vocabulary.
- Let Q, K, V be matrices in $\mathbb{R}^{d \times d}$
  - $\mathbf{q}_i = Q\mathbf{x}_i$
  - $\mathbf{k}_i = K\mathbf{x}_i$
  - $\mathbf{v}_i = V\mathbf{x}_i$

Output for each word is a weighted sum of values:

$$\mathbf{o}_i = \sum_j \text{softmax}_j(\mathbf{q}_i^T \mathbf{k}_j) \cdot \mathbf{v}_i$$

Self Attention: Three Problems

Problem	Solution
Encoder and decoder has no inherent notion of ordering. It's just a bag of words.	Add position representations to each token
Just a weighted average of a vector. No non-linearities.	Apply feedforward network to each self attention output
Decoder should not look into the future while training the predictor.	Mask out the future by setting attention weights to zero.

Self-attention

Fixing the sequence order problem

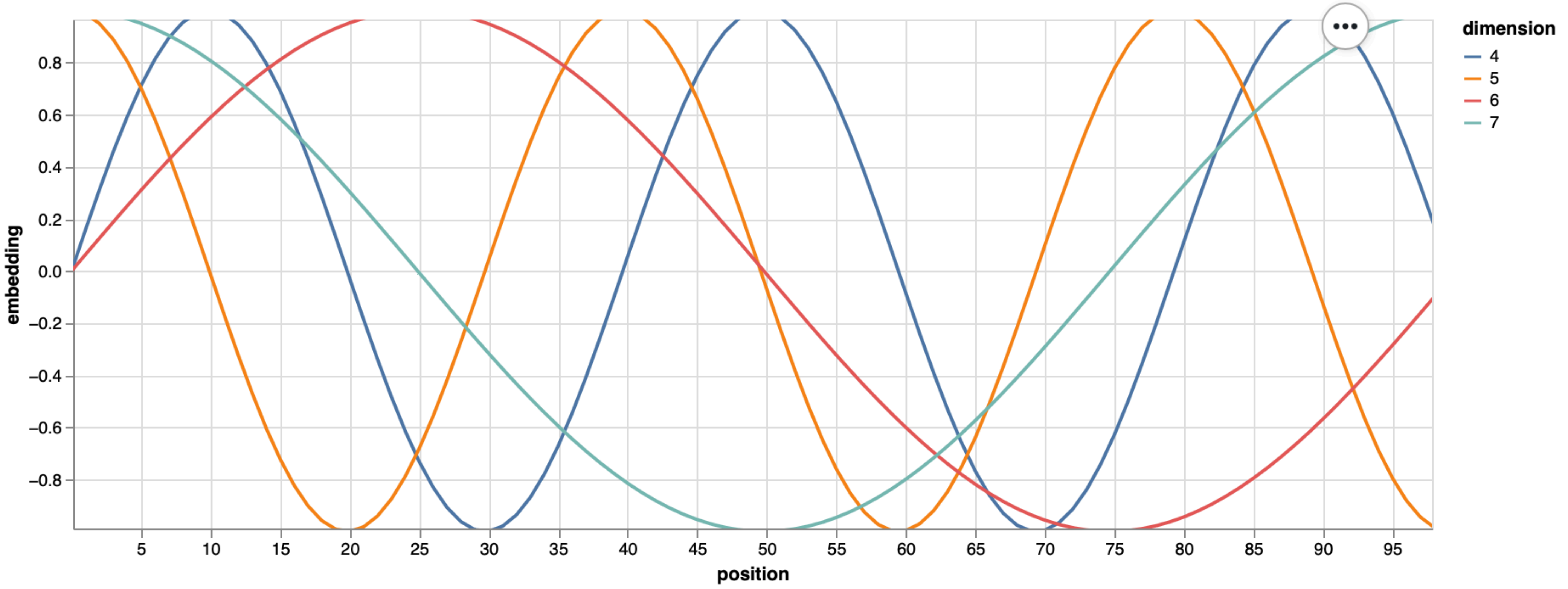
- We need to encode the order of the tokens in a sentence in the keys, values and queries
- We want a position embedding (similar to a word embedding)
- Let $\mathbf{p}_i \in \mathbb{R}^d$ for $i \in 1,\ldots,n$ be the position embeddings
- If $\mathbf{x}_i$ is the embedding for the word w_i then the combined word plus position embedding is $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$
- Either concatenate $\mathbf{x}_i$ and $\mathbf{p}_i$ or just add them. Adding is more common.

Position embeddings without learning

cs224n-self-attention-transformers-2023_draft.pdf

Use a periodic function like sine and cosine with different periods to get an embedding vector without any parameter updates.

$$\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



http://nlp.seas.harvard.edu/annotated-transformer/#positional-encoding

Pros:

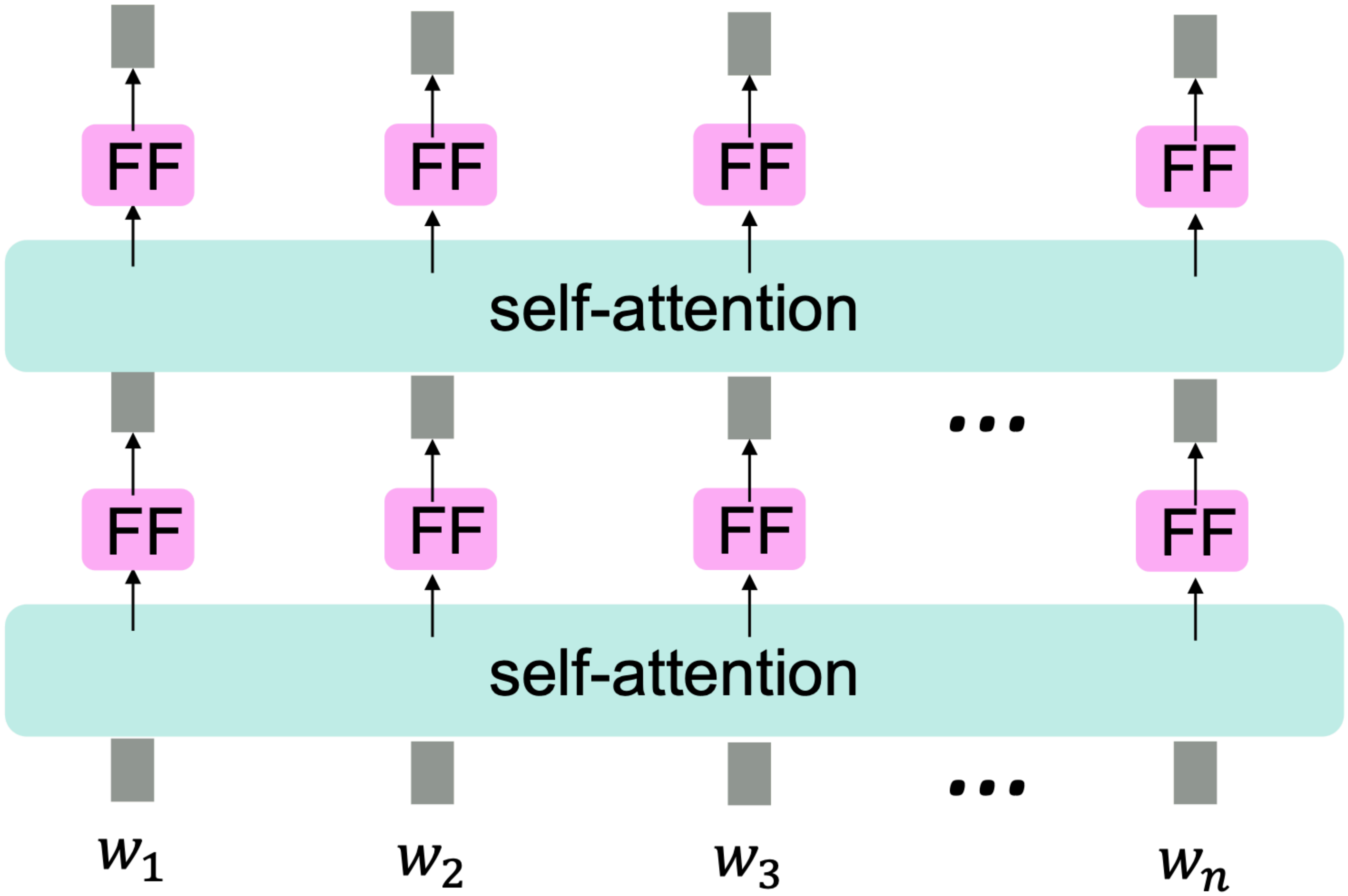
* Periodicity means absolute position is not important
* Can extrapolate to longer sequences as periods restart

Cons:

* Not learnable
* Extrapolation does not work that well for some applications

Self Attention Encoder using a Feed-forward Network

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \text{ output}_i + b_1) + b_2$$



Intuition: the feed-forward (FF) network processes the attention vector and makes it usable by the next layer

cs224n-self-attention-transformers-2023_draft.pdf

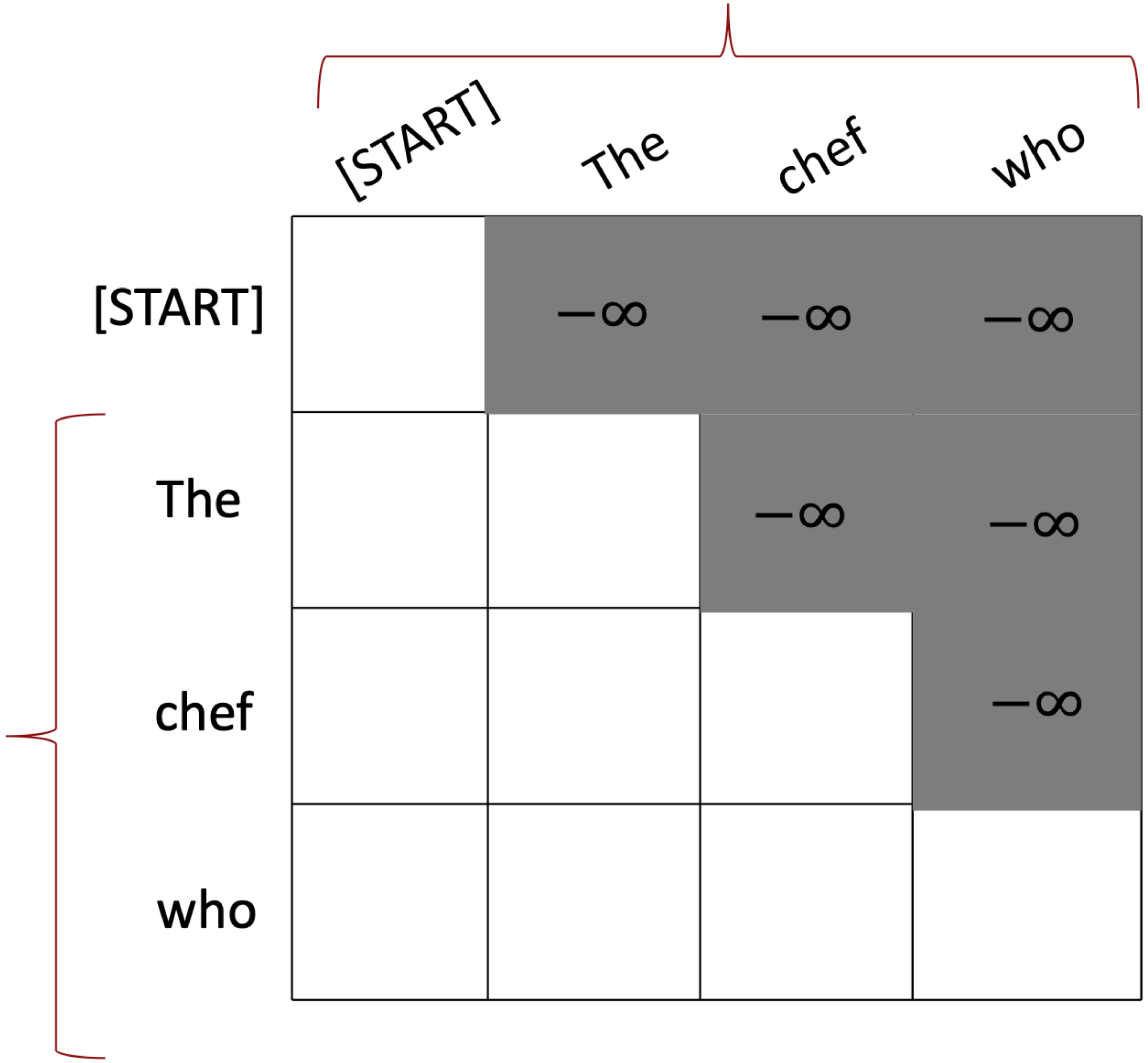
Decoders should **not** see into the future

We can only look at the non-greyed out words in the attention vector

* During training we mask the attention vector by setting attention scores to $-\infty$
* During inference, we decode from left to right and use the output from previous time-step as input to the next

$$e_{ij} = \begin{cases} q_i^\top k_j, j \leq i \\ -\infty, j > i \end{cases}$$

For encoding these words



cs224n-self-attention-transformers-2023_draft.pdf

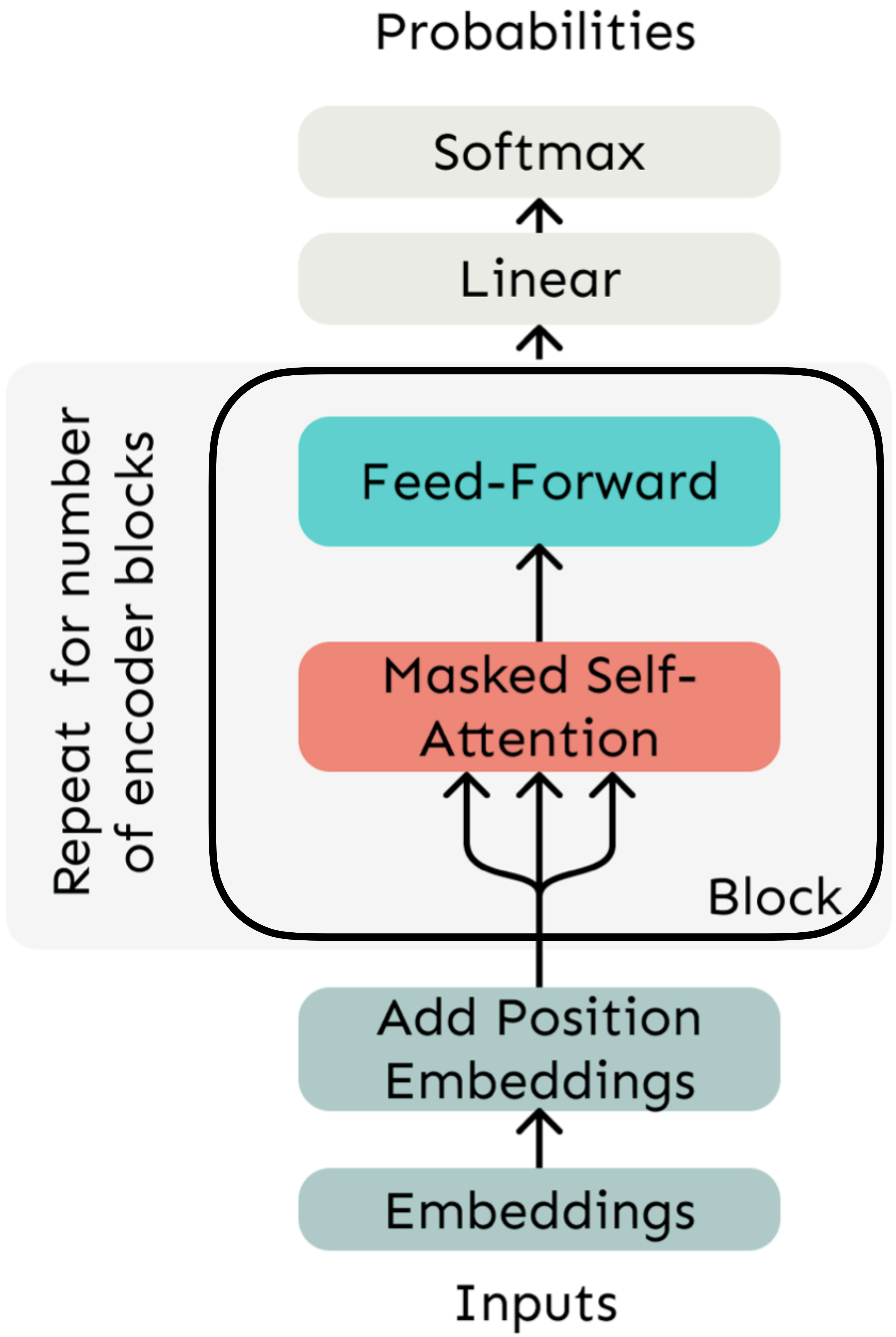
Self-attention building block

* **Self attention**
  * need this!

* **Position embeddings**
  * since self-attention is unordered

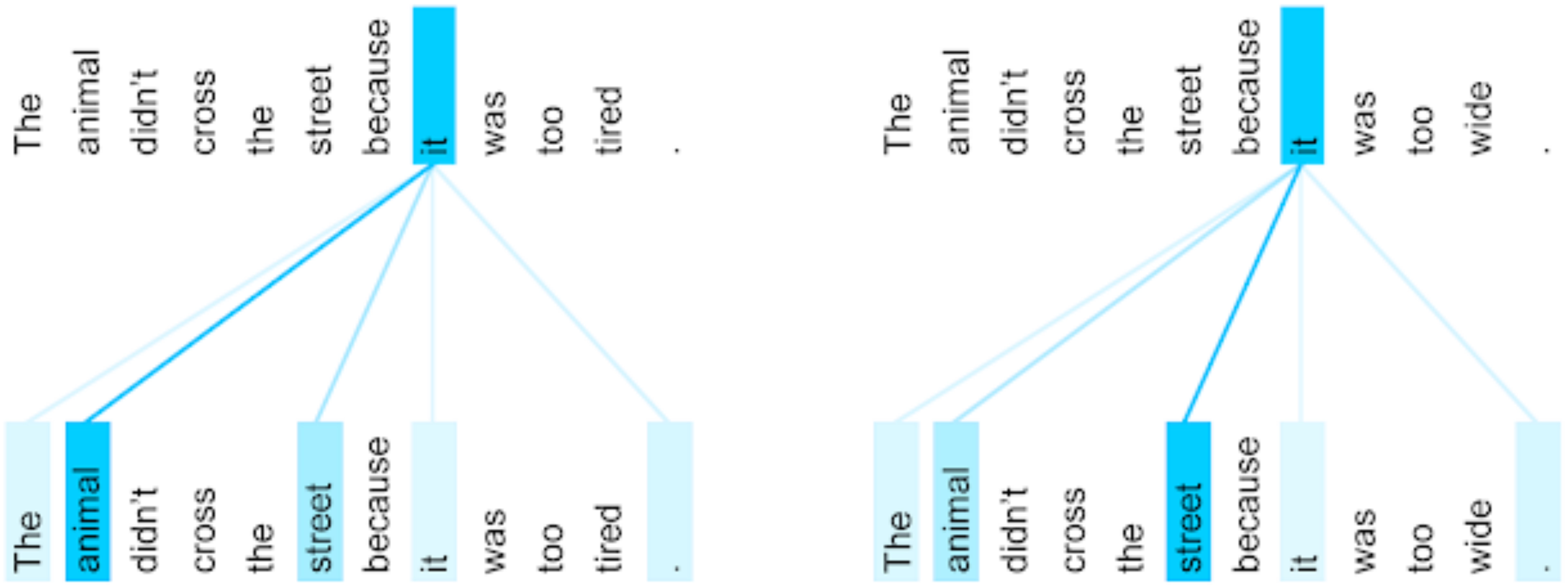
* **Nonlinearities**
  * For the output of attention block
  * Simple feed-forward network that is easy to train

* **Masking**
  * To parallelize operations while not looking at the future (during training)
  * Enforces training to behave like inference



cs224n-self-attention-transformers-2023_draft.pdf

From Single Attention Head to Multiple Attention Heads



https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Each Layer has Multi-head Self-Attention

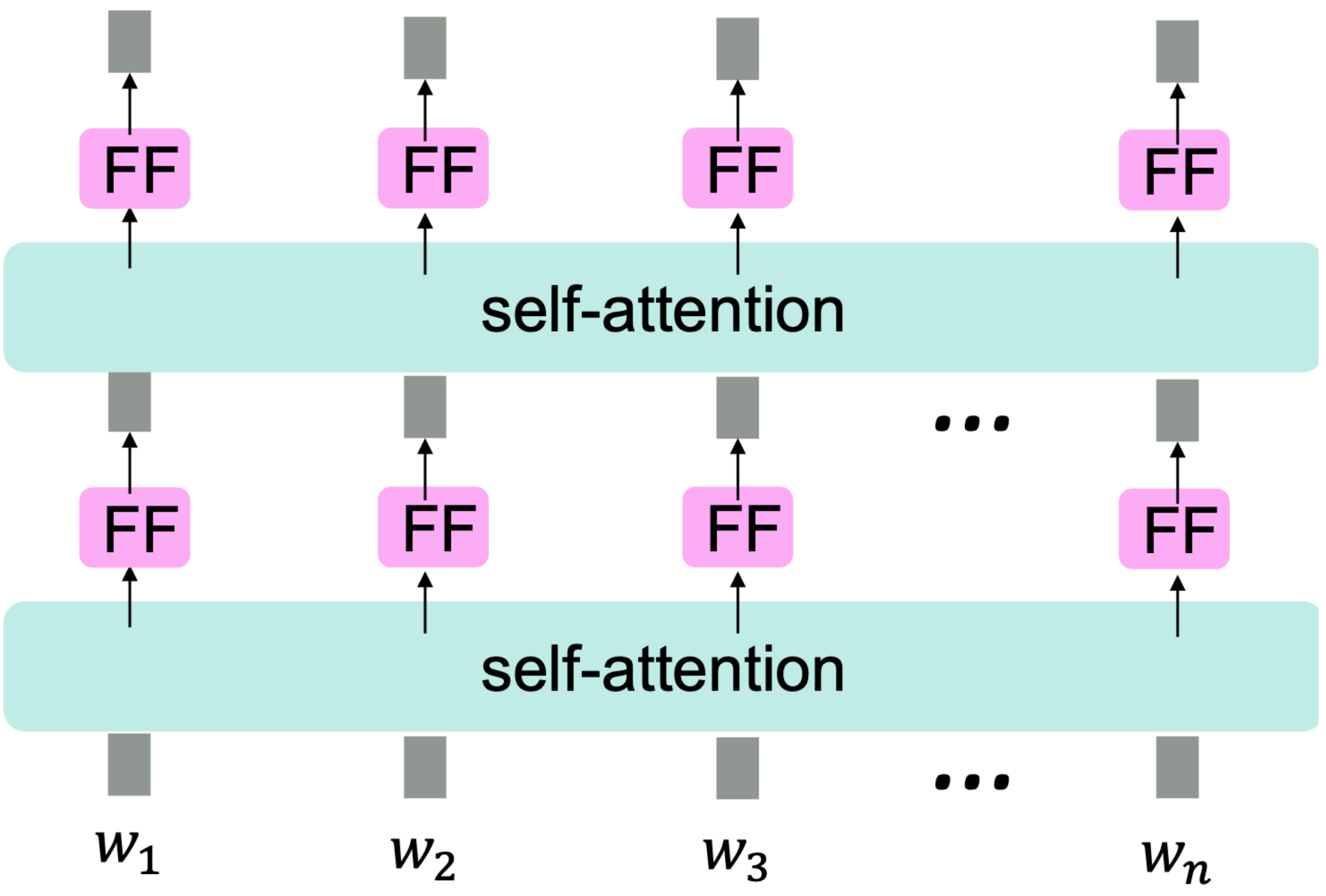
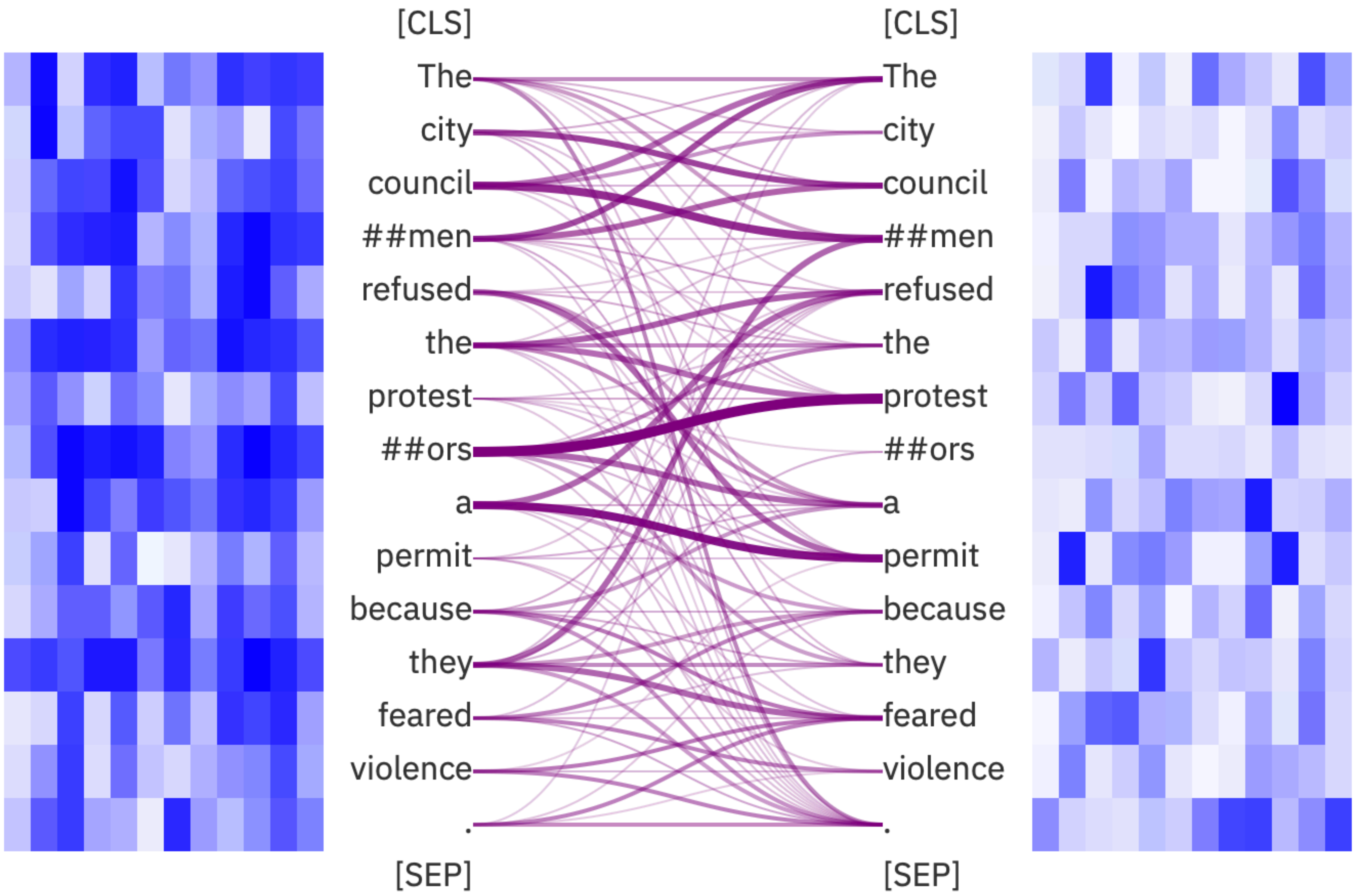


Image shows Layer 5 of a 12 Layer Transformer

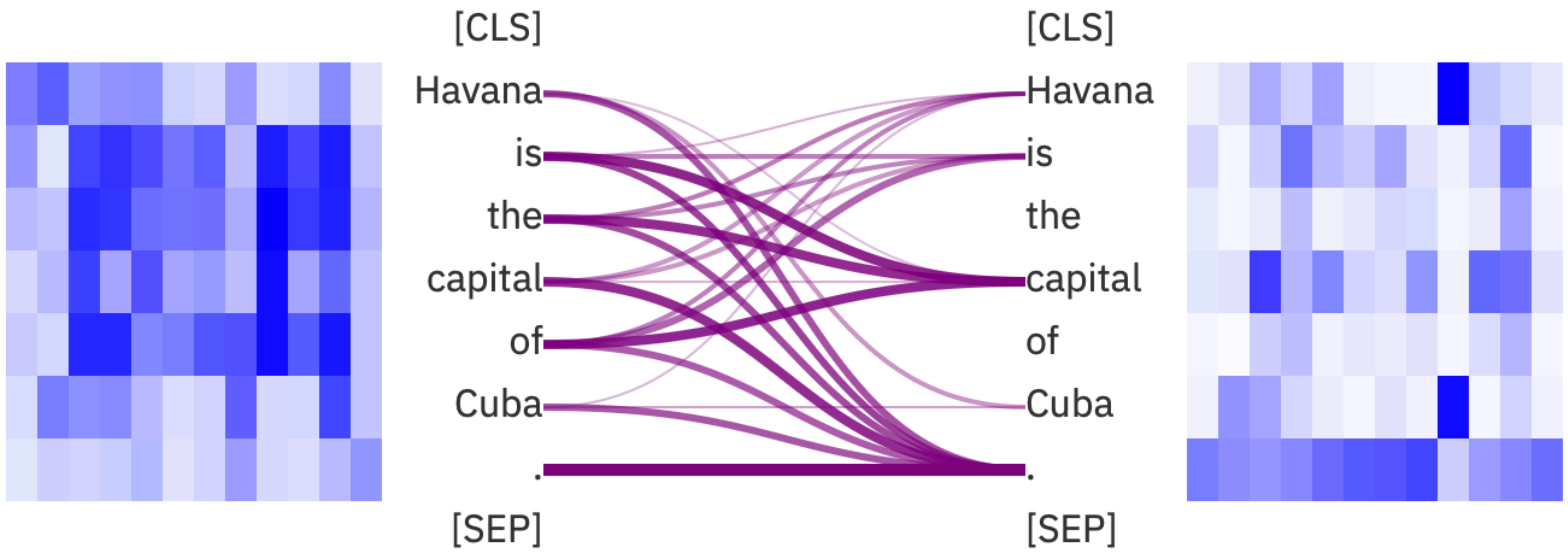
12 attention heads for each layer

https://huggingface.co/spaces/exbert-project/exbert



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

Selected heads: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12



https://huggingface.co/spaces/exbert-project/exbert

Self-attention

Matrix form

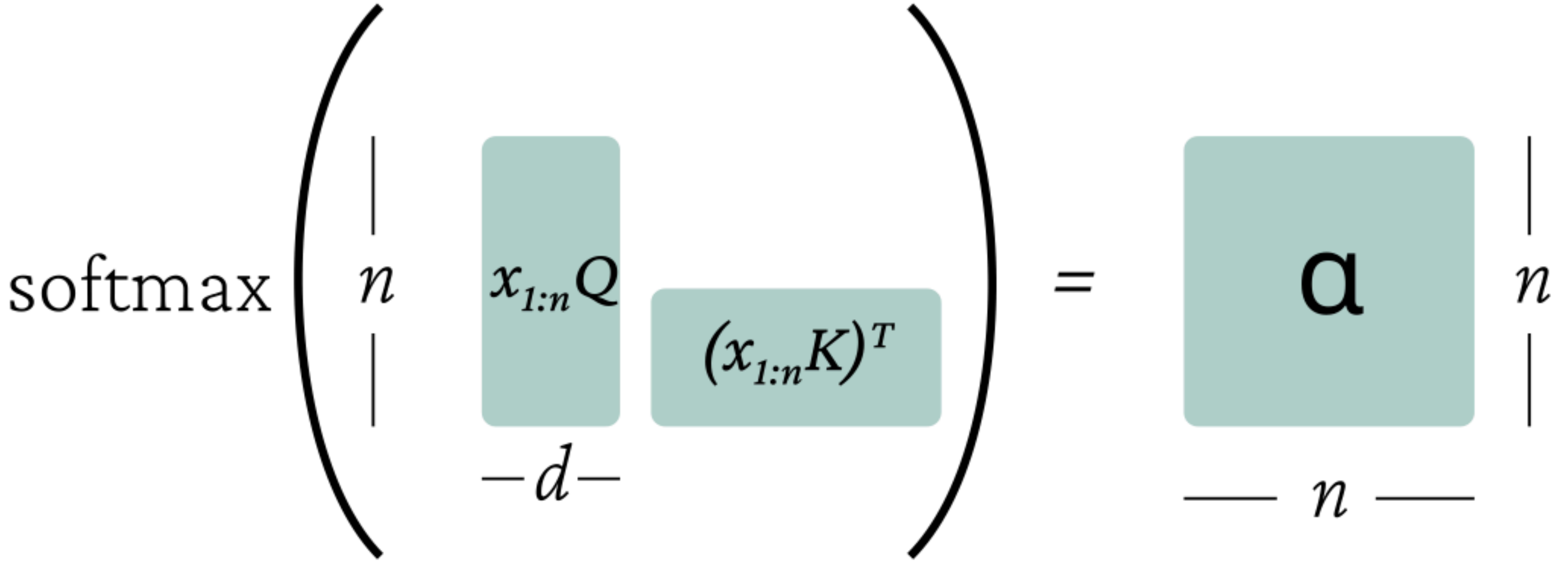
- Let $\mathbf{w} = (w_1, \ldots, w_n)$ be a sequence of tokens, like "Havana is the capital of"
- For each w_i let $\mathbf{x}_i = E\mathbf{w}_i$ where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix. V is the vocabulary.
- Let $\mathbf{X} = [\mathbf{x}_i; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of the input word vectors
- Let Q, K, V be matrices in $\mathbb{R}^{d \times d}$ then $XQ \in \mathbb{R}^{n \times d}, XK \in \mathbb{R}^{n \times d}, XV \in \mathbb{R}^{n \times d}$

cs224n-self-attention-transformers-2023_draft.pdf

Self-attention

Matrix form

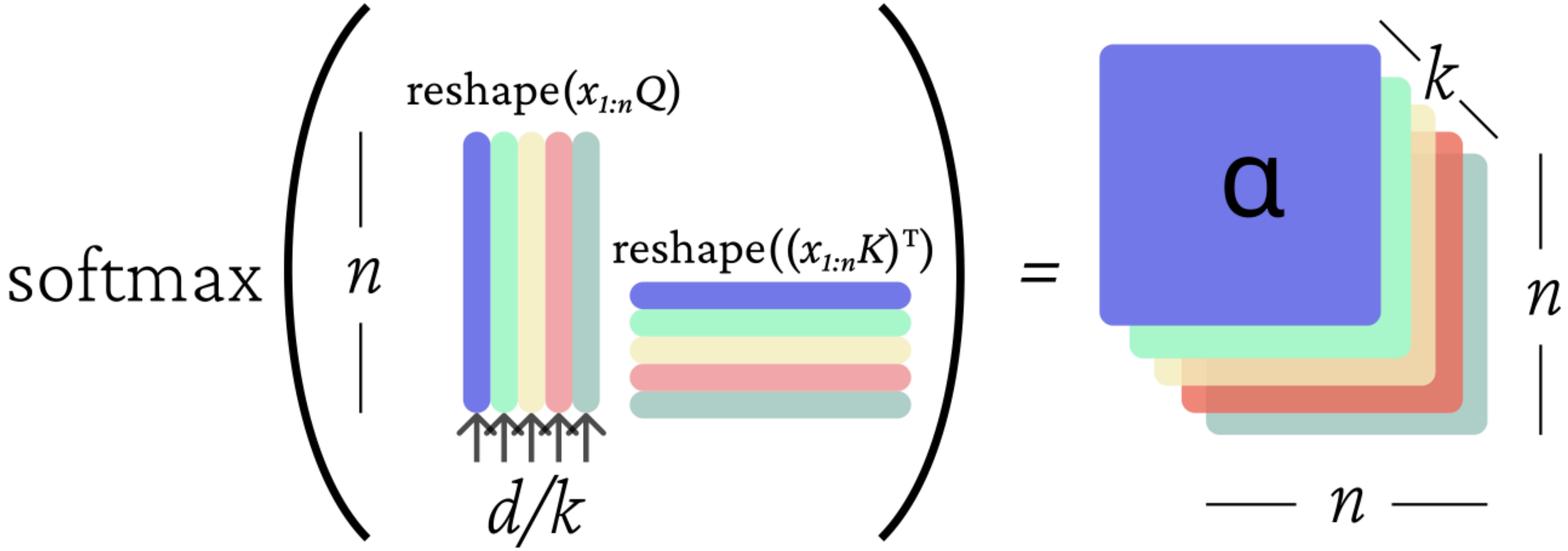
- First take the query-key dot products in matrix form: $XQ(XK)^T$
- Next softmax and compute the weighted average: *softmax*$(XQ(XK)^T)$ $\cdot XV \in \mathbb{R}^{n \times d}$
- Output is the context vector for each w_i but in matrix form: $\mathbb{R}^{n \times d}$



Multi-head Self-attention

Matrix form

- Let h range from $1 \ldots k$ for k total attention heads.
- $Q_h, K_h, V_h \in \mathbb{R}^{d \times \frac{d}{k}}$ so the output $O_h = \text{softmax}(XQ_h(XK_h)^T) \cdot XV_h \in \mathbb{R}^{\frac{d}{k}}$
- Combine all the heads: $O = [O_1, \ldots, O_k]$



Add & Norm

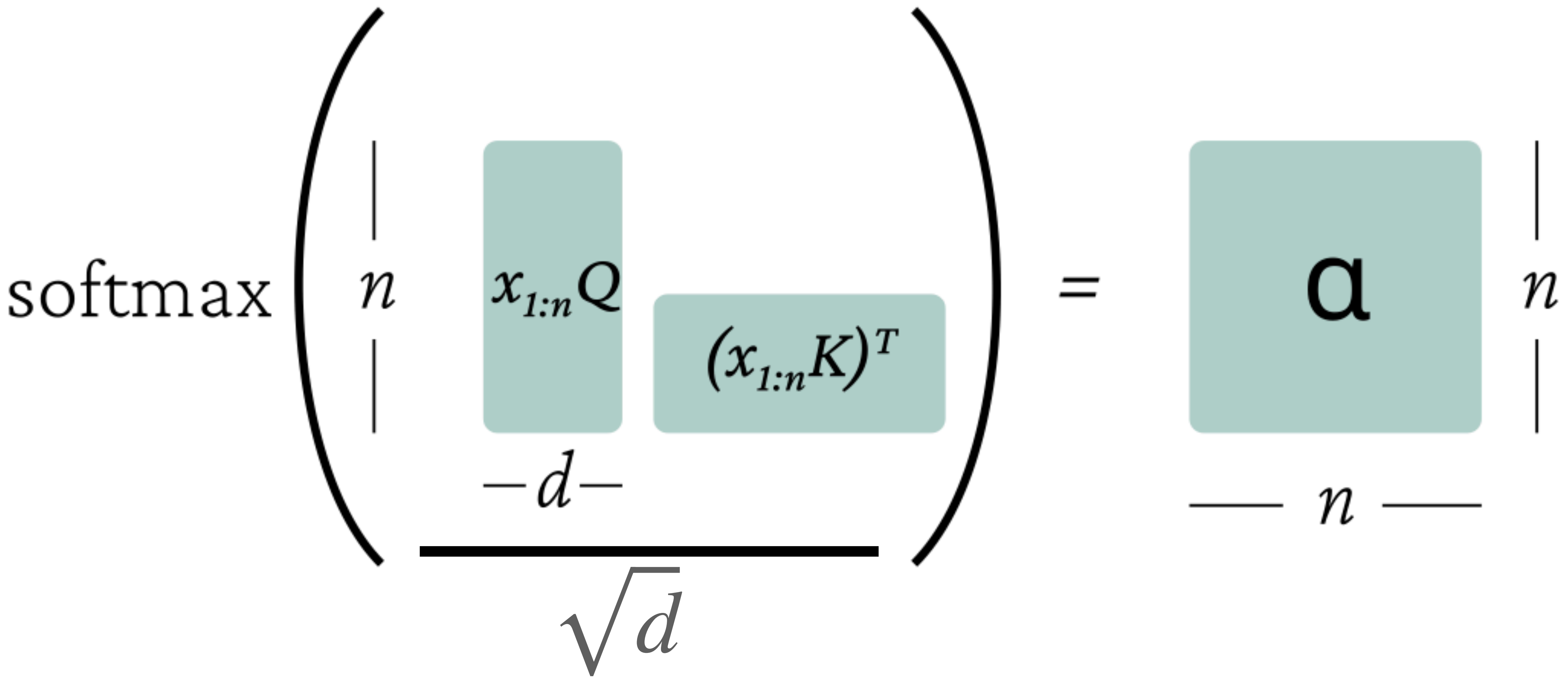
Residual Connections and Layer Norm

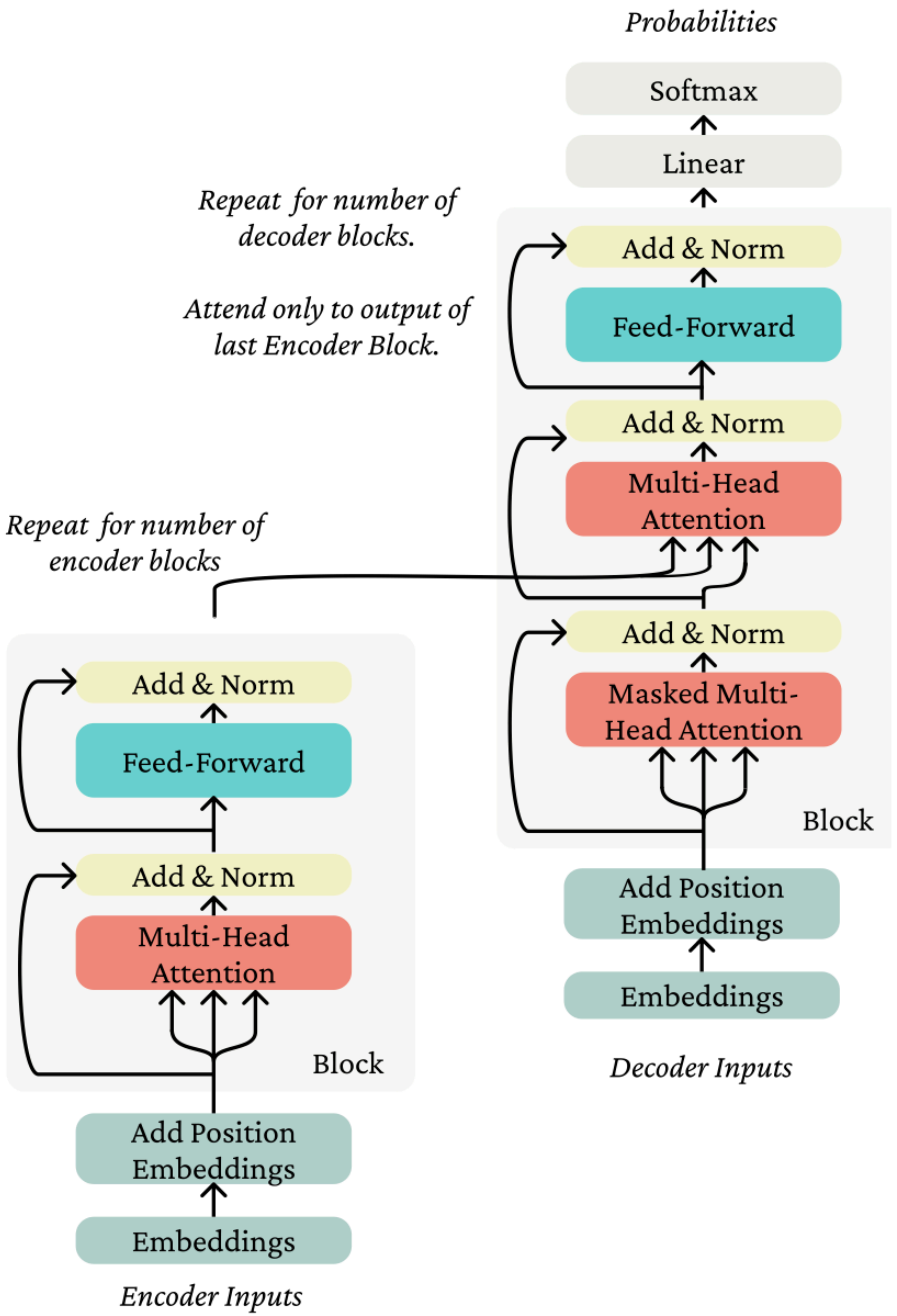
- Combine residual connection and layer norm into a single "Add & Norm" component
- Two choices:
  - Pre-norm: $\mathbf{z}^{\ell+1} = f(\text{LN}(\mathbf{z}^{\ell})) + \mathbf{z}^{\ell}$
  - Post-norm: $\mathbf{z}^{\ell+1} = \text{LN}(f(\mathbf{z}^{\ell}) + \mathbf{z}^{\ell})$
- Pre-norm leads to faster training. https://arxiv.org/abs/2002.04745

Scaled dot product attention

Attention with logit scaling

- Scaling to large dimension vectors d
- Dot product of random vectors (at initialization) grows proportional to $\sqrt{d}$
- Normalize the dot products by $\sqrt{d}$ to stop this iterative scaling upwards





Transformer Encoder-Decoder

cs224n-self-attention-transformers-2023_draft.pdf

Machine Translation Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Same Transformer model applied to constituency parsing

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Problems with Transformers

What needs fixing?

- Quadratic compute cost
  - In prior models like RNNs attention grew linearly since it only paid attention to the previous time step
  - In Transformers, attention takes $O(n^2 d)$ time to compute for input of length n and dimensionality d
- Positional representations
  - Simple indices the best we can do?
- In the decoder, attention at time t is independent of previous time steps.