
Dall-E

Word Representations and Attention Models

Applied Machine Learning
Derek Hoiem

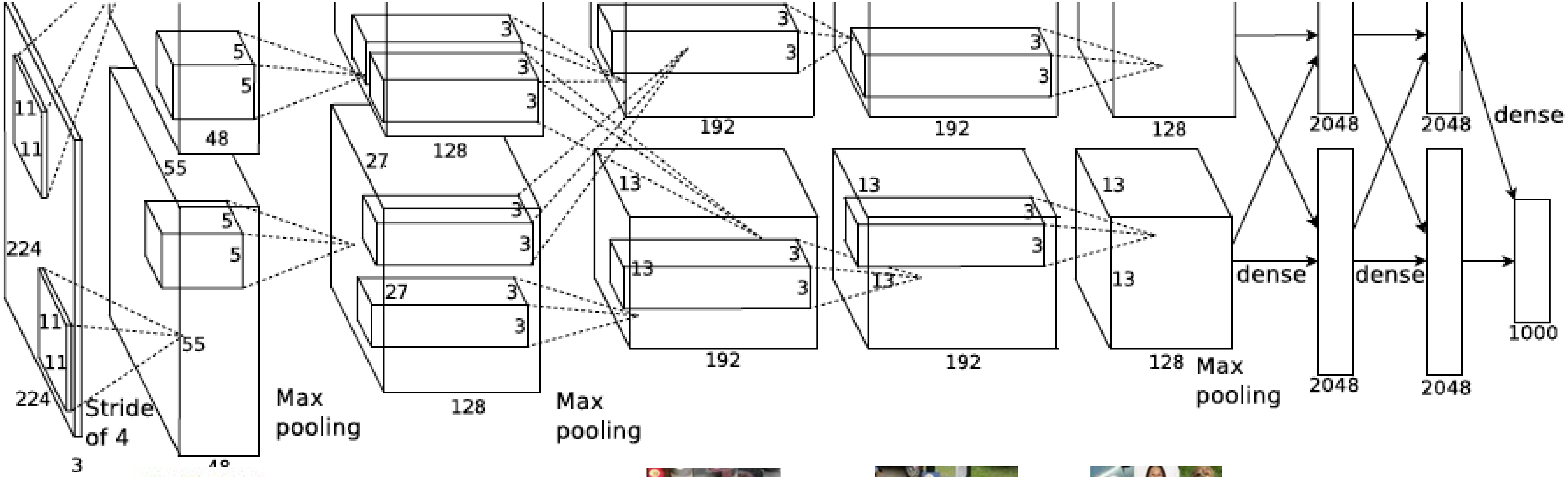
Today’s Lecture

- Representing natural language text as integers
  - Byte pair encoding
  - WordPiece
- Representing text tokens with continuous vectors
  - Word2Vec
- Masking Language Models and Attention
  - “Attention is all you need” transformers

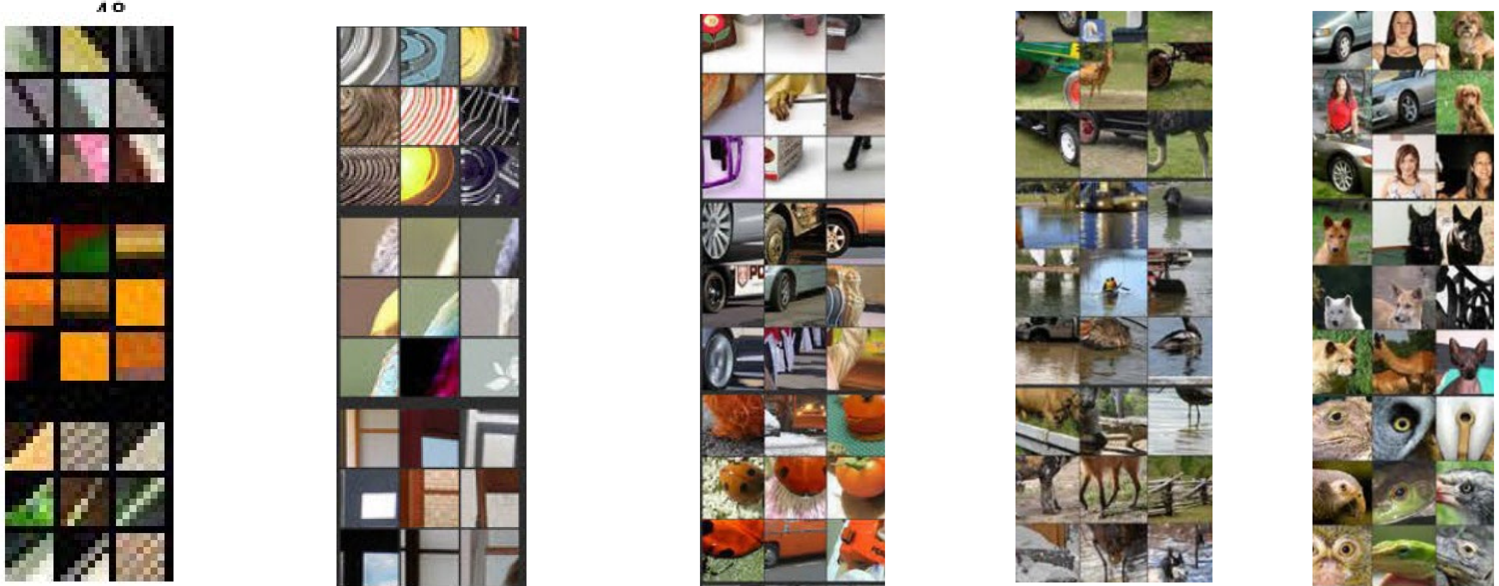
Each pixel means little, but images can be interpreted by grouping and recognizing patterns in groups of groups of groups of pixels

https://www.istockphoto.com/photos/funny-animal

CNNs iteratively process pixels->edges/colors->textures->sub-parts->parts->objects/scenes



Examples of strongly activating regions



But in text, the meaning is already in the words… right?

Which of these is more similar?

He sat on the chair, and it broke.

The chair says the department is broke.

After sitting, the seat is broken.

Which of these is more similar?

He sat on **the** **chair**, and it **broke**.

The **chair** says **the** department is **broke**.

After sitting, **the** seat is broken.

- Same word (character sequence) may mean different things
- Different words may mean similar things
- **Word meaning depends on surrounding words**

He sat on **the** **chair**, and it **broke**.

The **chair** says **the** department is **broke**.

After sitting, **the** seat is broken.

To analyze text, need to convert text to tokens

"**Token**": an integer or vector that represents a data element, a unit of processing

- With integer tokens, the values are not continuous (e.g. 5 is no closer to 10 than 5000)
- With vector tokens, similarity/distance (typically L2, dot product or cosine) is meaningful

Word → Integer

- Each unique space-delimited character string is assigned to a different integer
  - To limit vocabulary size, assign only the most frequent words to integers
  - Others are <unk> (unknown)
- Pros and cons
  - Simple
  - Possible to compare/retrieve documents based on count of tokens
  - Many words map to unknown (e.g. 1298, Bart's, Area-52, anachronism, …)
  - Large vocabulary needed
  - Does not model similarity of related words like broke/broken

He sat on **the** **chair**, and it **broke**.

The **chair** says **the** department is **broke**.

After sitting, **the** seat is broken.

Character → Integer

- Each character is assigned to a unique integer
- Pros and cons
  - Simple
  - Every document within alphabet can be fully modeled
  - Small vocabulary (< 100 integers needed for English)
  - Sometimes, similar words will have similar sequences (broke/n)
  - Count of tokens is not meaningful
  - Character sequences are long

Subword → Integer

- Common sequences of characters are assigned to unique integers
- Pros and cons
  - Every document within alphabet can be fully modeled
  - Vocabulary size is flexible (e.g. 30K for BERT, 50K for GPT-3)
  - Sometimes, similar words will have similar sequences (broke/n)
  - Need to solve for good subword tokenization

	Character	Subword	Word
"Chair is broken"	c, h, a, i, r, …	ch##, ##air, is, brok##, ##en	chair, is, broken
Vocabulary Size	**~ 100**	**4K-40K**	**> 30K**
Completeness	**Perfect**	**Perfect**	**Incomplete**
Independent Meaningfulness	**Bad**	**OK**	**Good**
Sequence Length	**Long**	**Medium** (e.g., 1.4 tokens per word)	**A little shorter**
Encodes word similarity	**Somewhat**	**A little better**	**Not at all**

Subword Tokenizers: Byte Pair Encoding

1. Start with each character assigned to a unique token
2. Iteratively assign a new token to the most common pair of consecutive tokens, until max_tokens is reached

Initial array of 4 characters

```
aaabdaaabac
```

Replace aa by Z

```
ZabdZabac
Z=aa
```

Replace ab by Y

```
ZYdZYac
Y=ab
Z=aa
```

Replace ZY by X

```
XdXac
X=ZY
Y=ab
Z=aa
```

```
XZd → ZYZd → aaabaad
```

Example from Wikipedia

WordPiece Tokenizer (Sennrich et al., Wu et al. 2016)

- **Word**: Jet makers feud over seat width with big orders at stake
- **wordpieces**: _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

r ·	→	r·
l o	→	lo
lo w	→	low
e r·	→	er·

Figure 1: BPE merge operations learned from dictionary {‘low’, ‘lowest’, ‘newer’, ‘wider’}.

For each merge:

1. Count token pair frequencies in dataset
2. Select most frequent pair
3. Merge that “best” pair
    a. Assign best pair to new token
    b. Replace all instances of best pair in dataset with that token

https://arxiv.org/abs/1508.07909
https://arxiv.org/pdf/1609.08144.pdf

Try it

Do first two merges of:

`Your cat cannot do the can-can, can he?`

For each merge:

1. Count token pair frequencies in dataset
2. Select most frequent pair
3. Merge that “best” pair
    a. Assign best pair to new token
    b. Replace all instances of best pair in dataset with that token

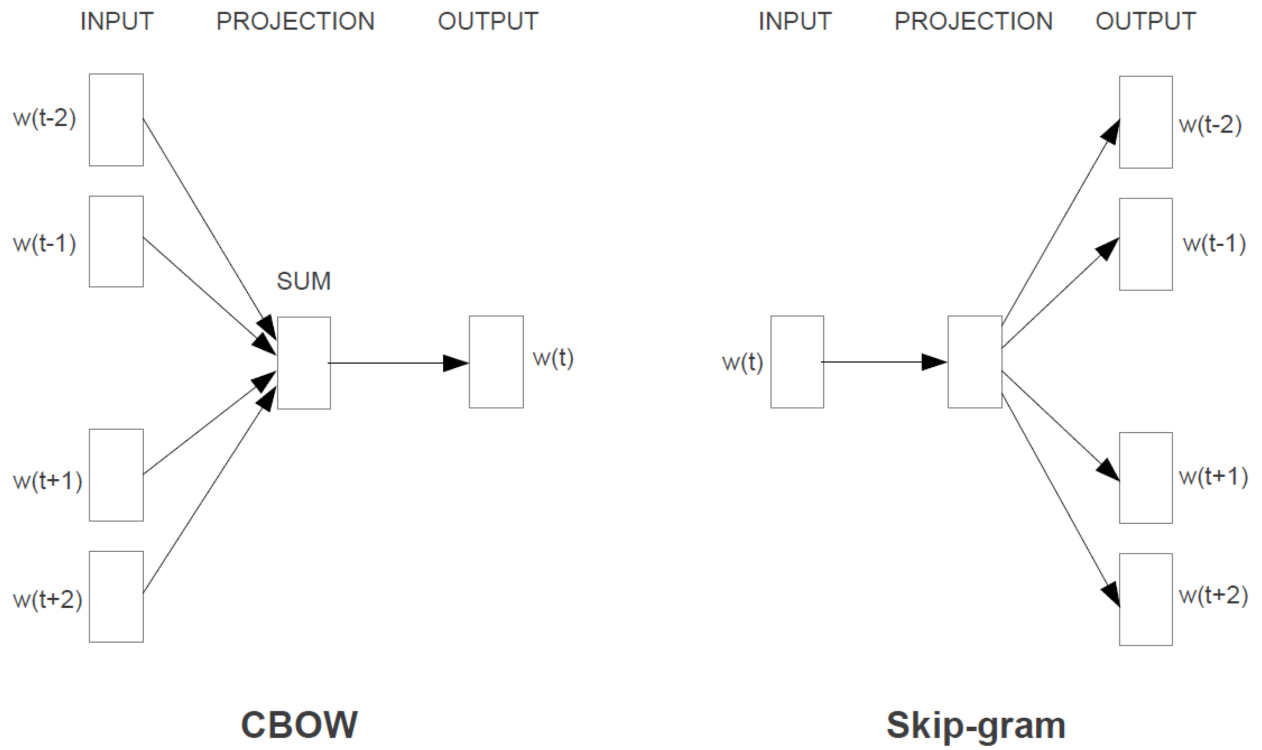
How can we better encode word similarity?

- Different words are related to each other
- Encode “meaning” in a continuous vector
- Learn these vectors based on surrounding words

Word2Vec (Mikolov et al. 2013)

For each word, solve for a continuous vector representation:

- CBOW: predict center word as average of surrounding words (after projecting each word to a vector)
- Skip-Gram: each word (after projecting to a vector) predicts each surrounding word with a linear model



https://arxiv.org/pdf/1301.3781.pdf

Train by gradient descent

- At the end, each word integer can be replaced by a fixed-length continuous vector
- These vectors can predict word relationships

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Word2Vec predicted relationship examples

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

E.g., Paris – France + Italy = Rome

Word2Vec demos

https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/

https://remykarem.github.io/word2vec-demo/

2 Minute Break: think of other relations to try

A new type of data processing

- Linear: output is sum of weights times features
- Convolution: output at each position is linear function of features within a window
- Attention: given a set of <key, value> pairs and a <query>, output is sum of values weighted by key-query similarity

Cross-Attention

<key *k*, value *v*>: a data element, in which key is used for matching and value is used to output

<query *q*>: used to match keys and accumulate values

$$out(q) = \left[\sum_i s(k_i, q) v_i\right] / \left[\sum_i s(k_i, q)\right]$$

Similarity of ith key and query   ith value   Make similarities sum to 1

Cross-attention simple example

$$out(q) = \left[\sum_i s(k_i, q) v_i\right] / \left[\sum_i s(k_i, q)\right]$$

$s(k,q) = 1/(k-q)^2$

<key,value> pairs : $\langle 1, 1\rangle, \langle 7, -1\rangle, \langle 5, -1\rangle$

query: 4

$out = \left(\frac{1}{3^2}\cdot 1 + \frac{1}{3^2}\cdot -1 + \frac{1}{1^2}\cdot -1\right) / \left(\frac{1}{3^2} + \frac{1}{3^2} + \frac{1}{1^2}\right)$

$= -0.818$

query: 0

$out = \left(\frac{1}{1^2}\cdot 1 + \frac{1}{6^2}\cdot(-1) + \frac{1}{4^2}\cdot(-1)\right) / \left(\frac{1}{1^2} + \frac{1}{6^2} + \frac{1}{4^2}\right)$

$= 0.834$

Self-attention

- Key=value
- Each key is also a query

$$S(k,q) = 1/(k-q)^2+1$$

in: 1, 7, 5

out: $\left(\frac{1}{1}\cdot 1 + \frac{1}{6^2+1}\cdot 7 + \frac{1}{4^2+1}\cdot 5\right) / \left(1 + \frac{1}{6^2+1} + \frac{1}{4^2+1}\right) = 1.37,$

$6.54, 5.13)$

apply again: $(1.76, 6.06, 5.29)$

apply again: $(2.19, 5.64, 5.42)$

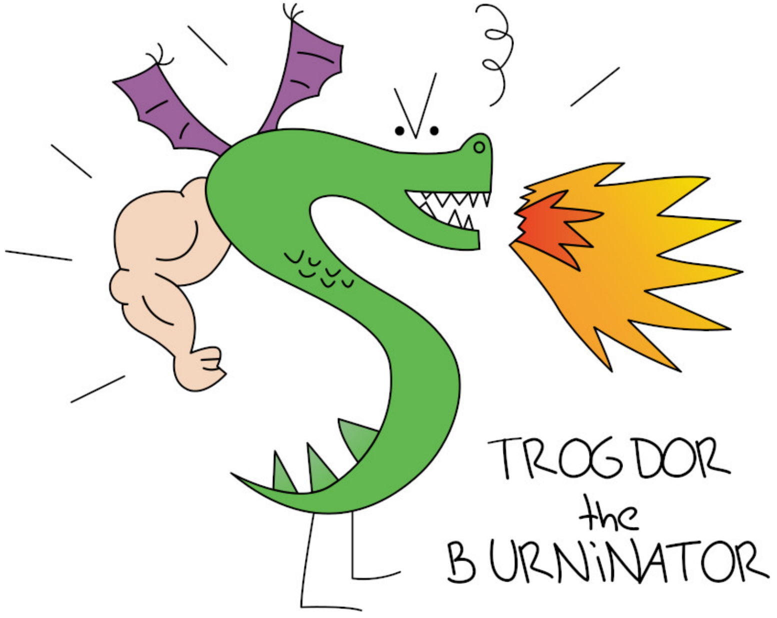
Another example of self attention

$$S(k,q) = 1/((k-q)^2+1)$$

Input (k,q,v)	iter 1	iter 2	iter 3	iter 4
1.000	1.497	1.818	1.988	2.147
9.000	8.503	8.182	8.012	7.853
8.000	8.128	8.141	8.010	7.853
2.000	1.872	1.859	1.990	2.147

Attention

- Cross-Attention is an instance-based regression
- Self-Attention is a soft clusterer/aggregator and can be stacked
  - Adding multi-dimensional vectors can overlay multiple types of information, not just blend or replace
- Attention becomes extremely powerful and general when combined with learned similarity and non-linear feature transformations!



https://homestarrunner.com/

Transformer (Vaswani et al. 2017)

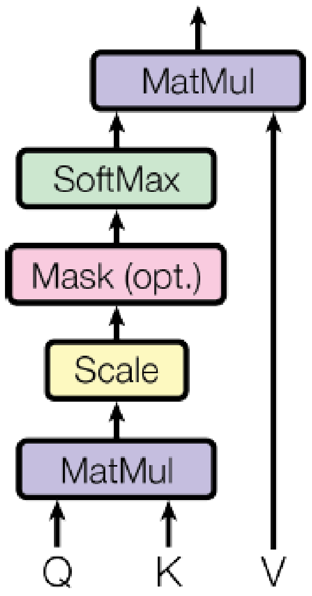
- Define similarity via linear projection with softmax

$$S(k_i, q) = \exp(k_i \cdot q)$$

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

Normalize by sqrt of dimensionality of keys

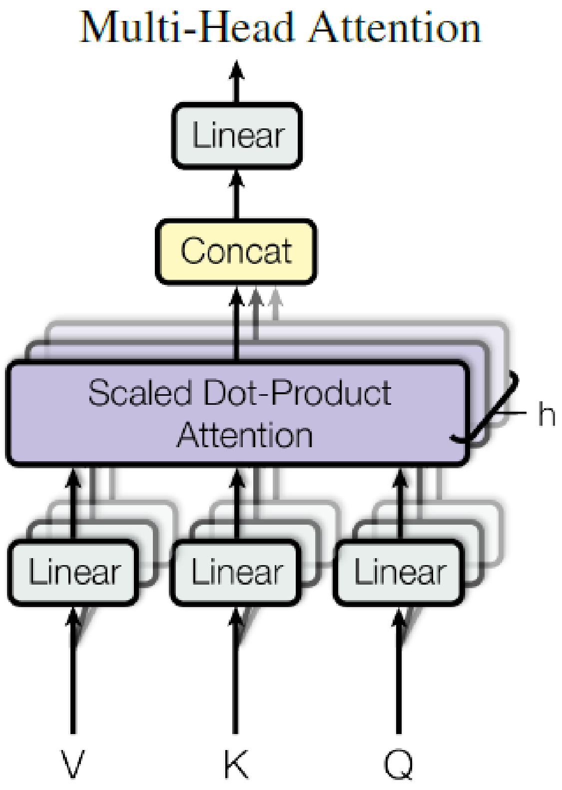
Scaled Dot-Product Attention



Attention is all you need

Transformer (Vaswani et al. 2017)

- One or more similarity functions can be learned with linear layers
  - If there are K similarities and D dimensions to input, each parallel linear layer outputs D/K values



$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1},\ldots,\mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

[Attention is all you need]

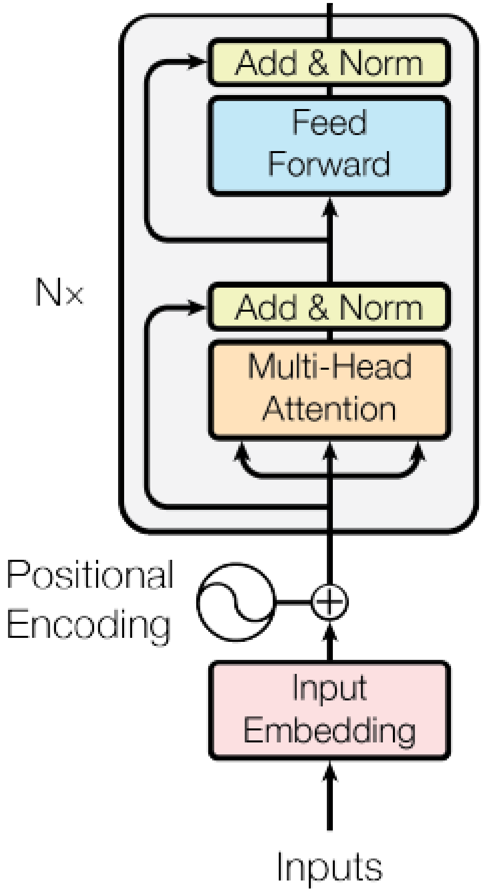
Transformers: general data processors

- **Input tokens can represent anything**: image patches, text tokens, audio, controls, etc.

- Invariant to order of tokens: **add positional embedding** to distinguish pos/type of input

- **Transformer block**:
    - Apply multi-head attention
    - Apply 2-layer MLP with ReLU to each token separately
    - Residual and layer norm (over all tokens) after each

- Can stack any number of transformer blocks

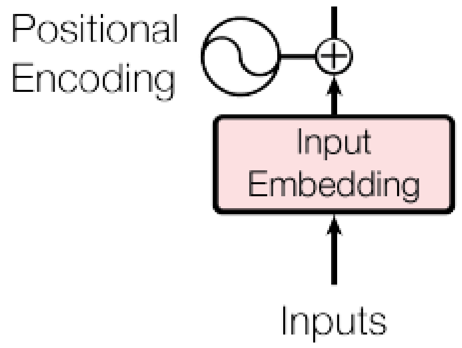


[Attention is all you need](#)

Positional encodings

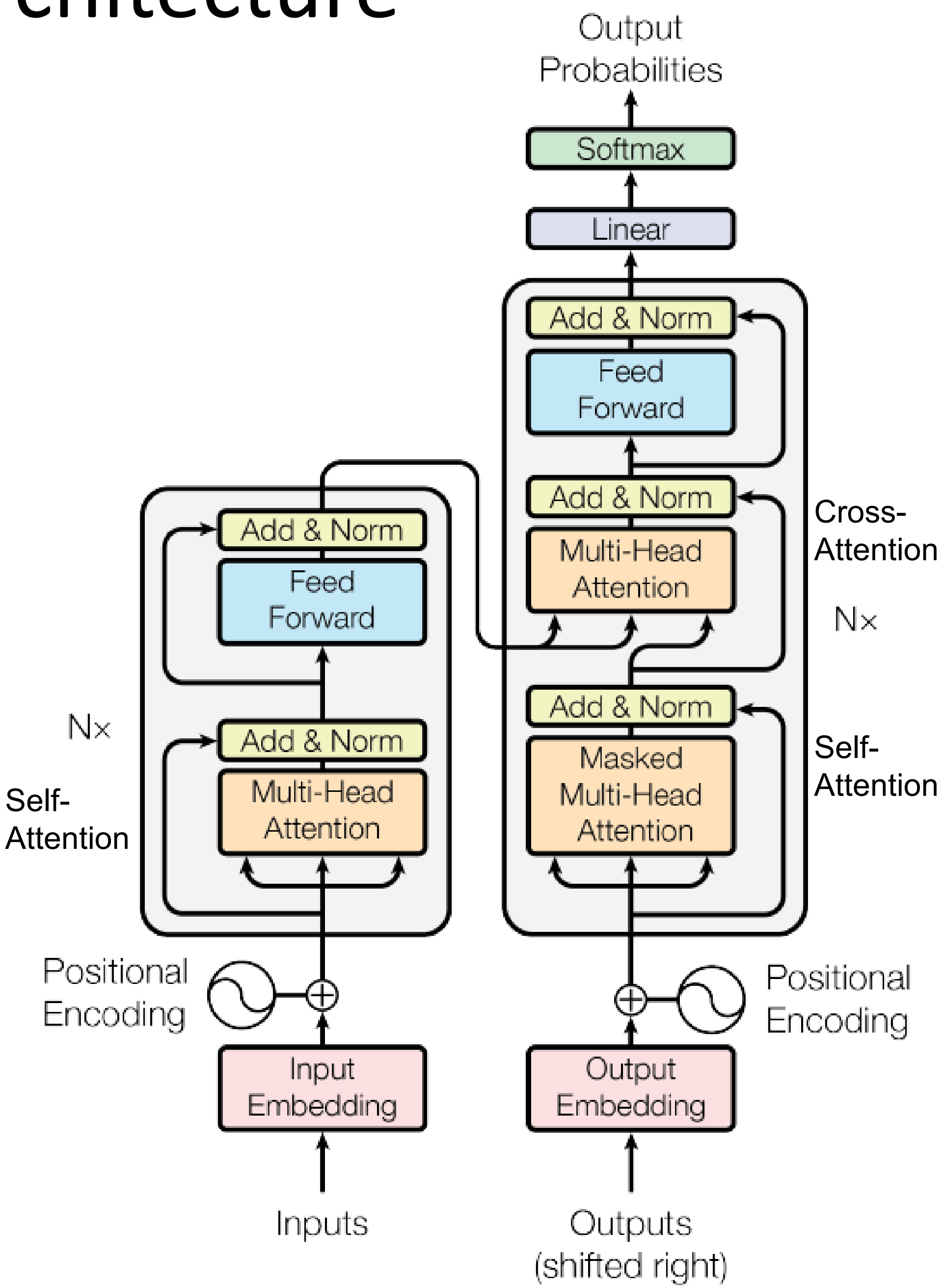
- Transformer processing does not depend on position of token
  - This is kind of similar to convolution, as each "patch" or token vector is processed independently, but no overlap between patches
  - But to compare between tokens, relative position may be important
- Sinusoidal encoding (on right) is such that a dot product between encodings corresponds to positional similarity
- Learned or even fixed random encodings also work similarly in practice

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Language Transformer: Complete Architecture

- WordPiece tokens (integers) are mapped to learned 512-d vectors
- Positional encoding added to each vector
- N=6 transformer blocks applied to input
- Until <EOS> is output:
  - Process input + output so far
  - Output most likely word (after more attention blocks and linear model)



[Attention is all you need](#)

Attention Visualizations

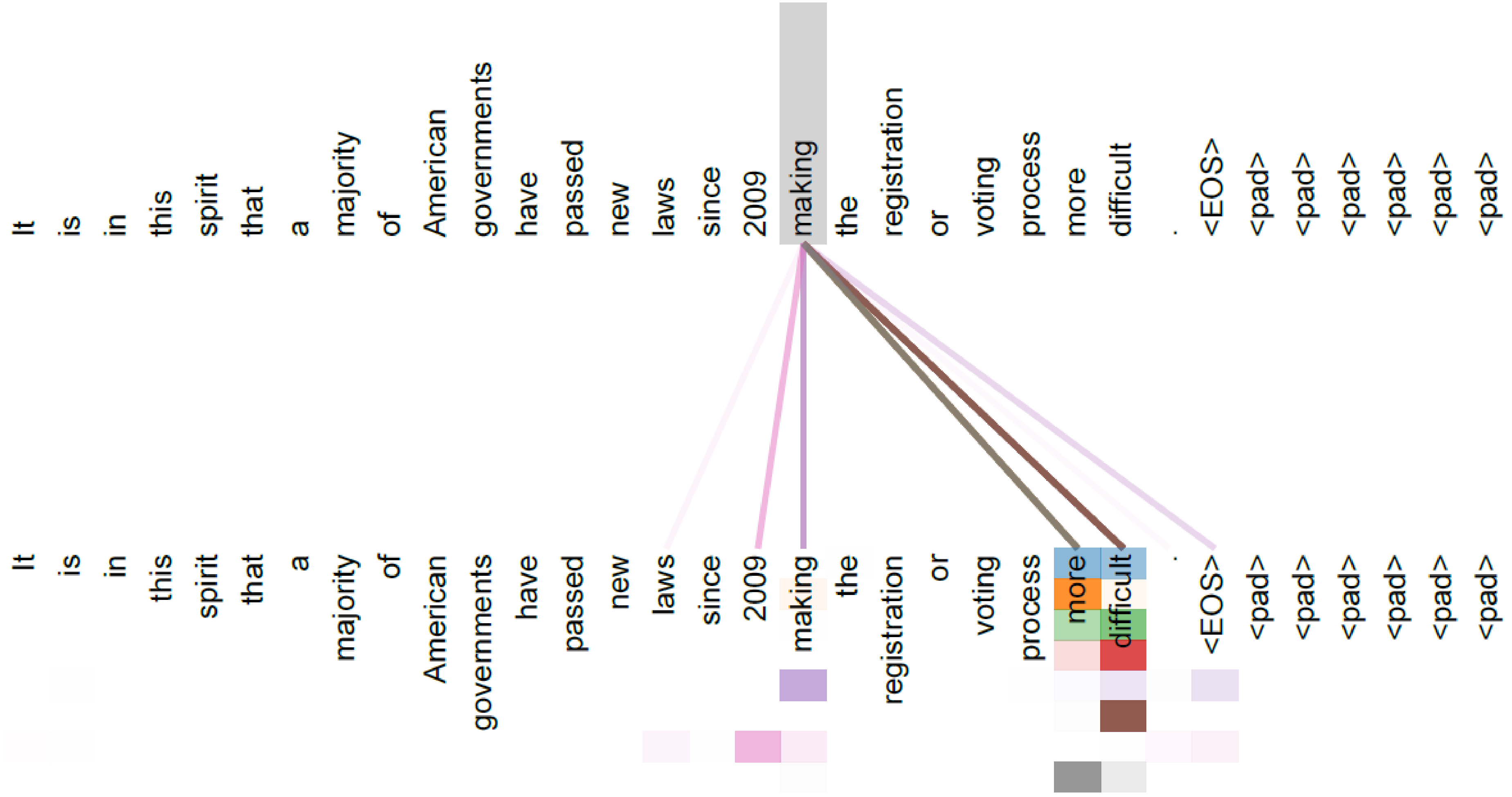


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

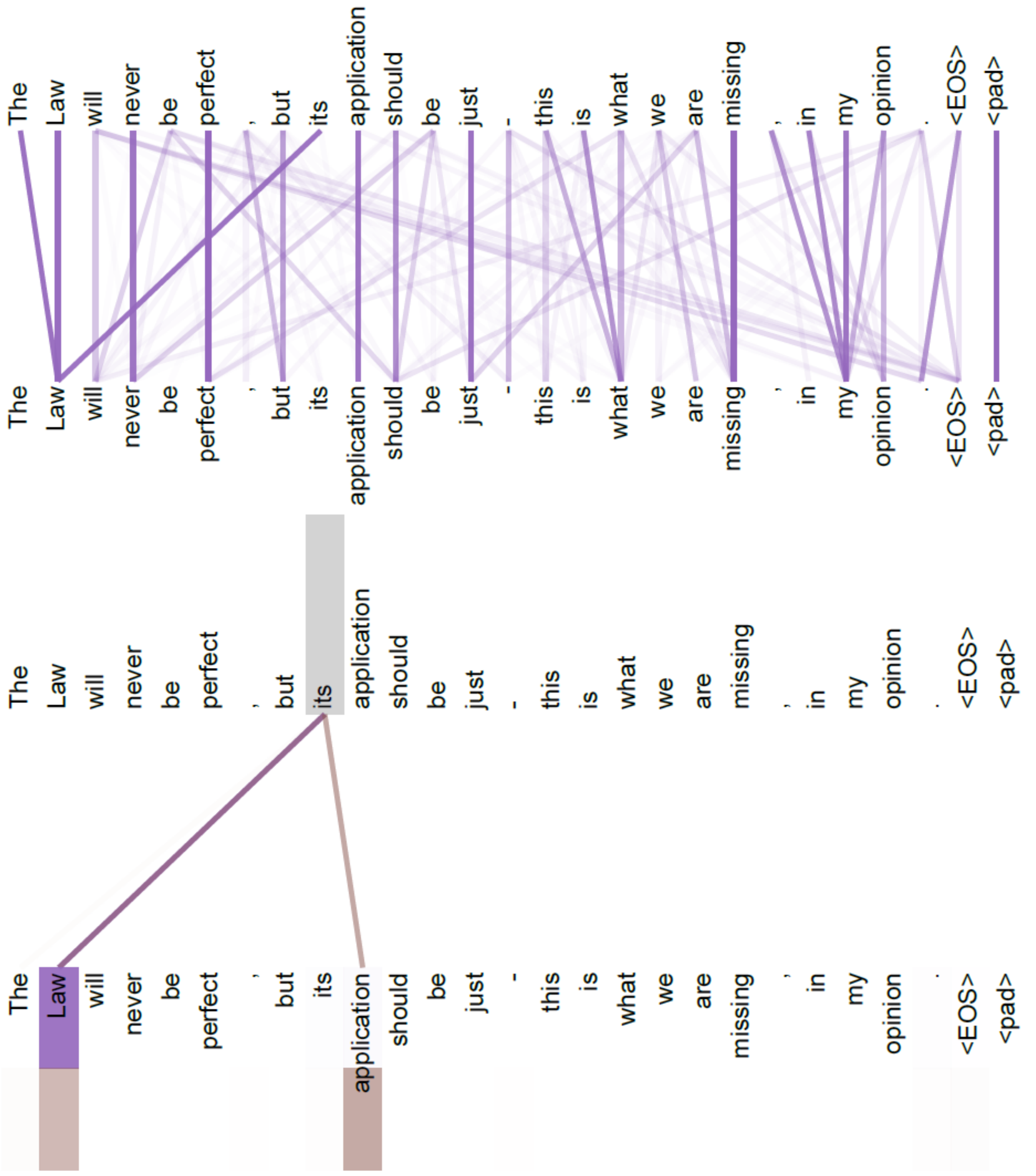


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

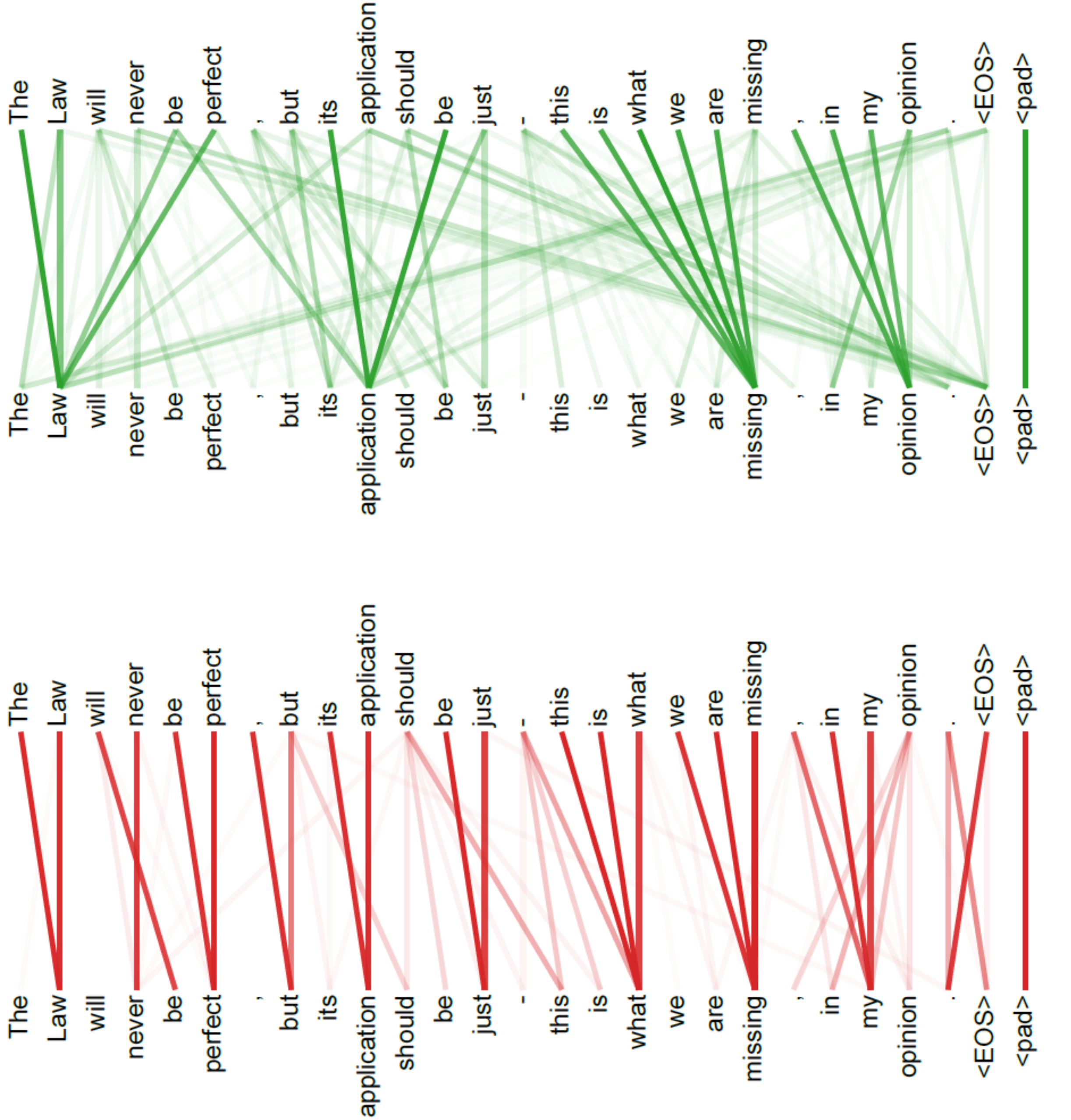
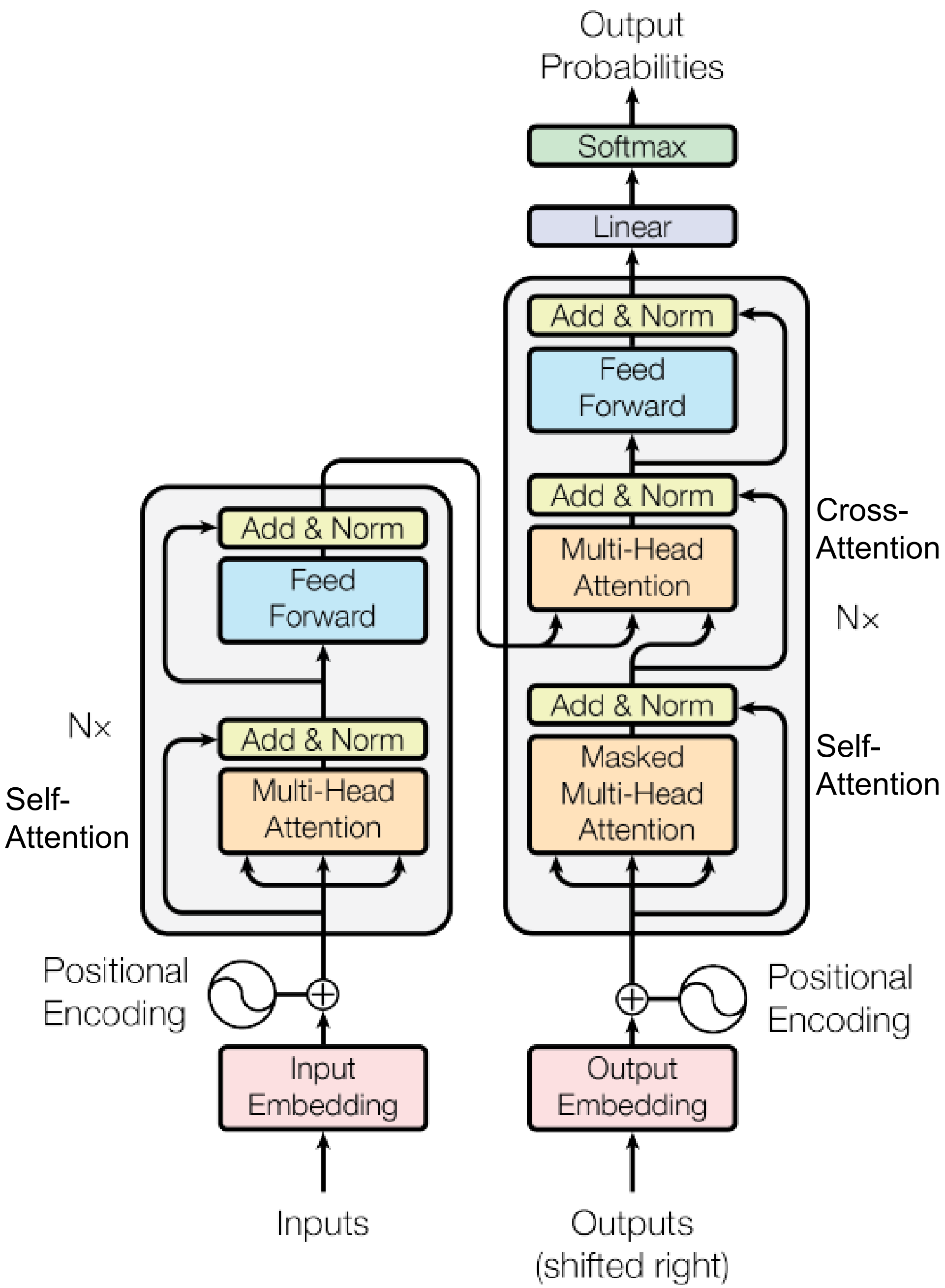


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Application to Translation

- English-German
  - 4.5M sentence pairs
  - 37K tokens
- English-French
  - 36M sentences
  - 32K tokens
- Base models trained on 8 P100s for 12 hours
- Big models (2x token dim, 3x training steps) trained for 3.5 days
- Adam optimizer: learning rate ramps up for 4K iterations, then down
- Regularization: drop-out, L2 weight, label smoothing



[Attention is all you need](#)

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**	$2.3 \cdot 10^{19}$	

Things to remember

Sub-word tokenization based on byte-pair encoding is an effective way to turn natural text into a sequence of integers

Chair is broken →
ch##, ##air, is, brok##, ##en

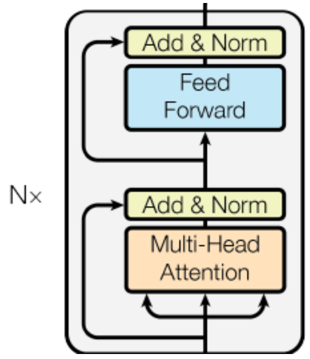
Learned vector embeddings of these integers model the relationships between words

Paris – France + Italy = Rome

Attention is a general processing mechanism that regresses or clusters values

Input (k,q,v)	iter 1	iter 2	iter 3	iter 4
1.000	1.497	1.818	1.988	2.147
9.000	8.503	8.182	8.012	7.853
8.000	8.128	8.141	8.010	7.853
2.000	1.872	1.859	1.990	2.147

Stacked transformer blocks are a powerful network architecture that alternates attention and MLPs



Further reading: http://nlp.seas.harvard.edu/annotated-transformer/

Next class: Transformers in Language and Vision

- BERT
- ViT
- Unified-IO