



UNIVERSITY
OF AMSTERDAM



INSTITUTE FOR LOGIC,
LANGUAGE AND COMPUTATION

Attention & Transformers

Ivo Verhoeven

i.o.verhoeven@uva.nl

14-04-2023

Attention & Transformers are deep topics, you should do some reading outside of this lecture

Intro

Blogs:

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

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

<https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/>

<https://theaisummer.com/topics/attention/>

Tutorials:

Phillip Lippe's DL1 tutorials

<https://theaisummer.com/einsum-attention/>

<https://jaykmody.com/blog/gpt-from-scratch/>

Attention & Transformers are deep topics, you should do some reading outside of this lecture

Intro

The original blog post from Google about Transformers. Sparse, but has some cool visuals:

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

Attention & Transformers are deep topics, you should do some reading outside of this lecture

Intro

The famous 'Illustrated Transformer' blog. Tons of great visuals, a step-by-step guide to self-attention, and additional blogs on other transformer architectures

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

Attention & Transformers are deep topics, you should do some reading outside of this lecture

Intro

The Lil'Log, contains great articles on many ML and NLP topics, with great visuals and high-level concepts. Maybe the best machine learning blog out there? Make sure to check out:

1. 'Attention? Attention!'
2. 'The Transformer Family (Version 2.0)'
3. 'Meta-Learning: Learning to Learn Fast' (not relevant for this topic, but maybe the course)

<https://lilianweng.github.io/>

Attention & Transformers are deep topics, you should do some reading outside of this lecture

Intro

‘AI Summer’ has many good blog posts (by various authors) of varying depth. Two excellent and relevant ones:

1. ‘Why multi-head self attention works: math, intuitions and 10+1 hidden insights’
2. ‘Understanding einsum for Deep learning: implement a transformer with multi-head self-attention from scratch’
 - Teaches you about einsum (best way to do tensor manipulation)
 - Teaches you to code up attention and transformers from scratch, which is easier than you think

<https://theaisummer.com/self-attention/>

Table of contents

Attention	A recipe for attention mechanisms, important properties and some connections
Examples	A whirlwind tour of attention mechanisms as found throughout NLP and ML applications
Transformers	Deep dive on the Transformer architecture: embeddings, self-attention layers, encoder-decoder stacks and vocabulary generation
Tips & Tricks	Necessary tricks of the trade when training and fine-tuning transformers

Attention is learned aggregation of information based on similarities of embeddings

Attention: Functional Form

$$\text{out} = \text{softmax} (f(\mathbf{Q}, \mathbf{K})) \mathbf{V}$$

Attention is learned aggregation of information based on similarities of embeddings

Attention: Functional Form

Four key ingredients:

1. Value: a tensor \mathbf{V} [$N \times T_V \times D_V$] of T_V elements, containing content to be aggregated

2. Key: a tensor \mathbf{K} [$N \times T_V \times D_Q$] of T_V elements, containing information *about* content

3. Query: a tensor \mathbf{Q} [$N \times T_Q \times D_Q$] of T_Q elements, containing information about information

4. Scoring function: a function f expressing similarity between elements in \mathbf{K} and \mathbf{Q}

$$\begin{aligned} \text{out} &= \sum_{t=1}^{T_V} \text{softmax}(f(\mathbf{Q}, \mathbf{k}_t)) \cdot \mathbf{v}_t \\ &= \text{softmax}(f(\mathbf{Q}, \mathbf{K})) \mathbf{V} \end{aligned}$$

Attention is learned aggregation of information based on similarities of embeddings

Attention: Functional Form

Softmax normalized, thus $0 < \alpha_t < 1$ (equality holding when infinities are output from f)

The tensor α [$N \times T_Q \times T_V$] is of the right dimensionality to transform the value tensor (of length T_V) to an aggregated tensor of length T_Q

$$\begin{aligned} \text{out} &= \sum_{t=1}^{T_V} \text{softmax}(f(\mathbf{Q}, \mathbf{k}_t)) \cdot \mathbf{v}_t \\ &= \underbrace{\text{softmax}(f(\mathbf{Q}, \mathbf{K}))}_{\alpha} \mathbf{V} \\ &\quad \text{Attention weights} \end{aligned}$$

Attention is essentially a learned form of pooling, out is now a tensor of [$N \times T_Q \times D_V$]

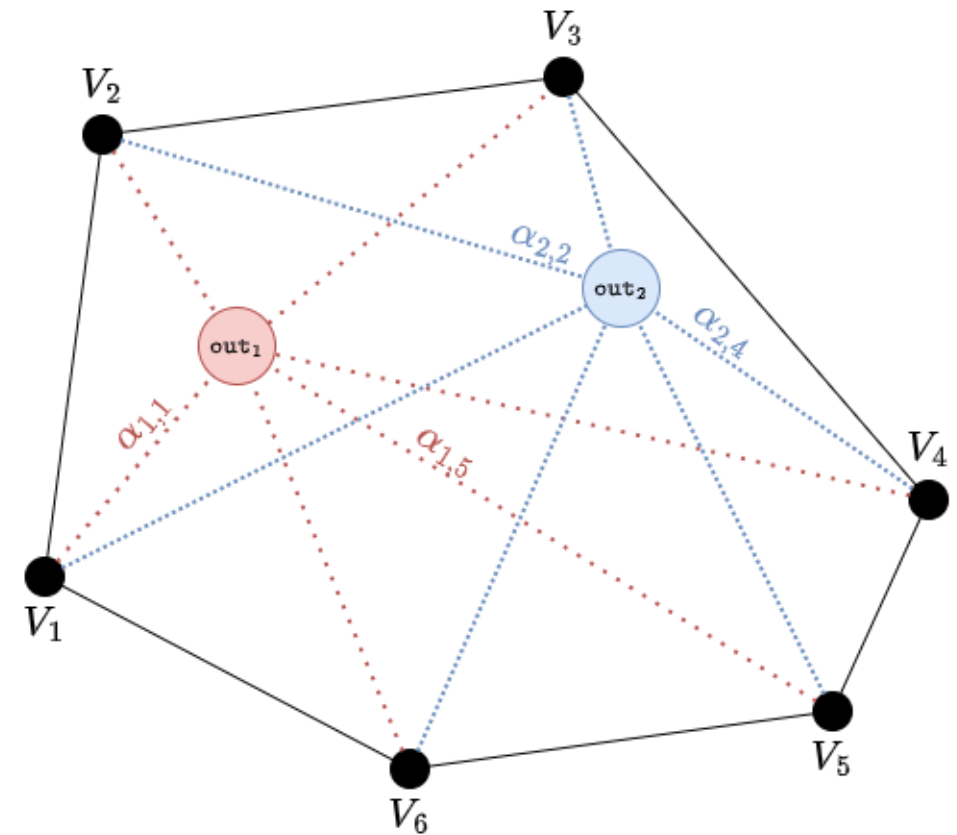
Attention is an operation over sets, not sequences

Attention: Properties

We can view the tensor $\alpha [N \times T_Q \times T_V]$ as a tensor containing T_Q convex combinations of the T_V elements in the value tensor \mathbf{V}

Attention, in essence, is nothing more than a **convex combination of the elements in the set of V**

The use of 'set' is deliberate



Attention is an operation over sets, not sequences

Attention: Properties

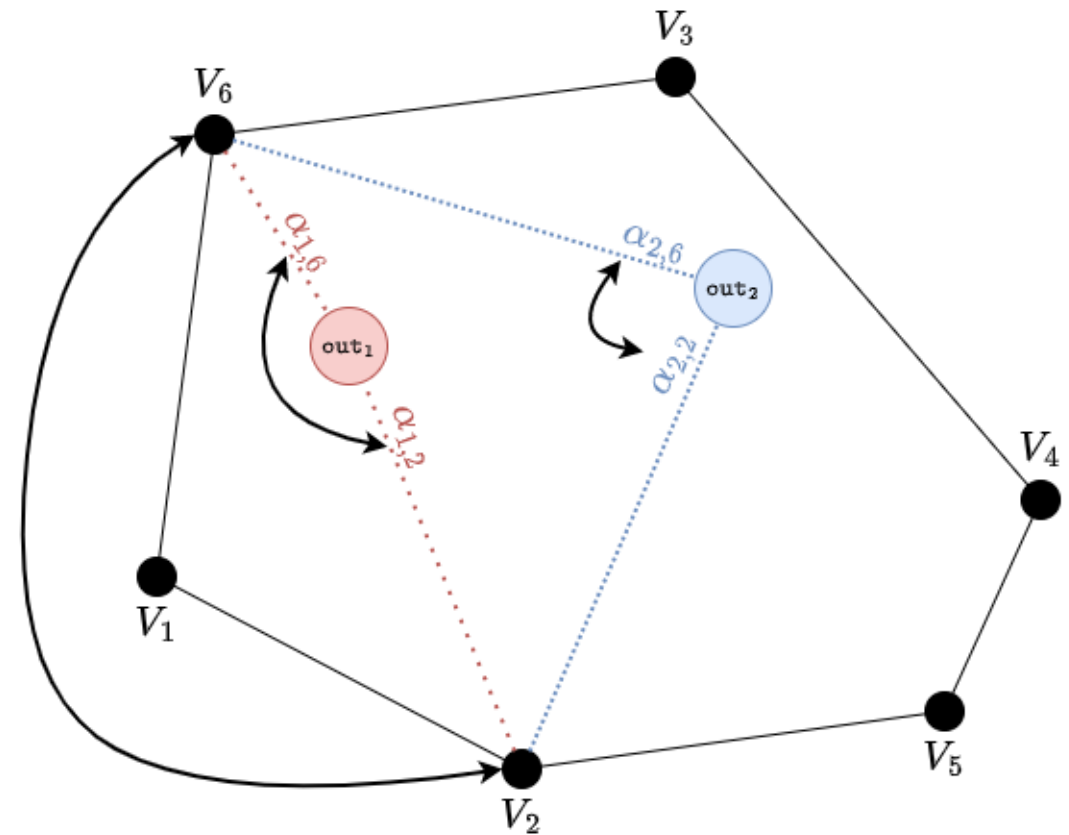
Attention is **permutation equivariant (in Q/V)**:

- Changing the order of the elements just means the output is equally re-ordered
- The aggregation is left untouched

If query is value-side data independent, attention becomes permutation *invariant* (in V)

Great property for graph neural networks

Less ideal for modelling language



Attention is a set operation that aggregates sequences based on similarities of embeddings

Attention: Analogy & Worked Examples

Despite an abstract definition, attention is actually very intuitive

Attention is... retrieval

Attention: Analogy & Worked Examples

A useful analogy is that of database retrieval,
e.g. in a library

1. **Value:** book titles
2. **Key:** topics/Dewey decimal system
3. **Query:** relevant topics in search
4. **Scoring function:** string matching between
key and query
























0	1	2	3	4	5	6
Information 209,306 	Philosophy and Psychology 219,109 	Religions 549,431 	Social sciences 1,138,734 	Language 166,708 	Natural sciences and mathematics 365,518 	Technology and Application of Knowledge 816,338 
00	01	02	03	04	05	06
Computing and Information 118,303 	Bibliographies 22,233 	Library and Information Sciences 26,570 	Dictionaries and Encyclopedias 10,026 	Special Topics 573 	Periodicals 4,377 T	Organizations 4,536 
000	001	002	003	004	005	006
General Works 783 	Knowledge 15,413 	History of the book 1,321 	Systems Theory 2,875 	Computer science 26,148 	Computer programming, programs, data, security 50,925 	Special Topics 20,534 
006.0	006.1	006.2	006.3	006.4	006.5	006.6
15	---	Special-purpose systems 131	Artificial Intelligence 8,348 	Pattern And Speech Recognition 785 	Digital audio 270	Computer Graphics 5,194
006.30	006.31	006.32	006.33	006.34	006.35	006.36
134	Machine Learning 1,038	Neural Networks 221	Knowledge-based Systems 336		Natural Language Processing 280 	1

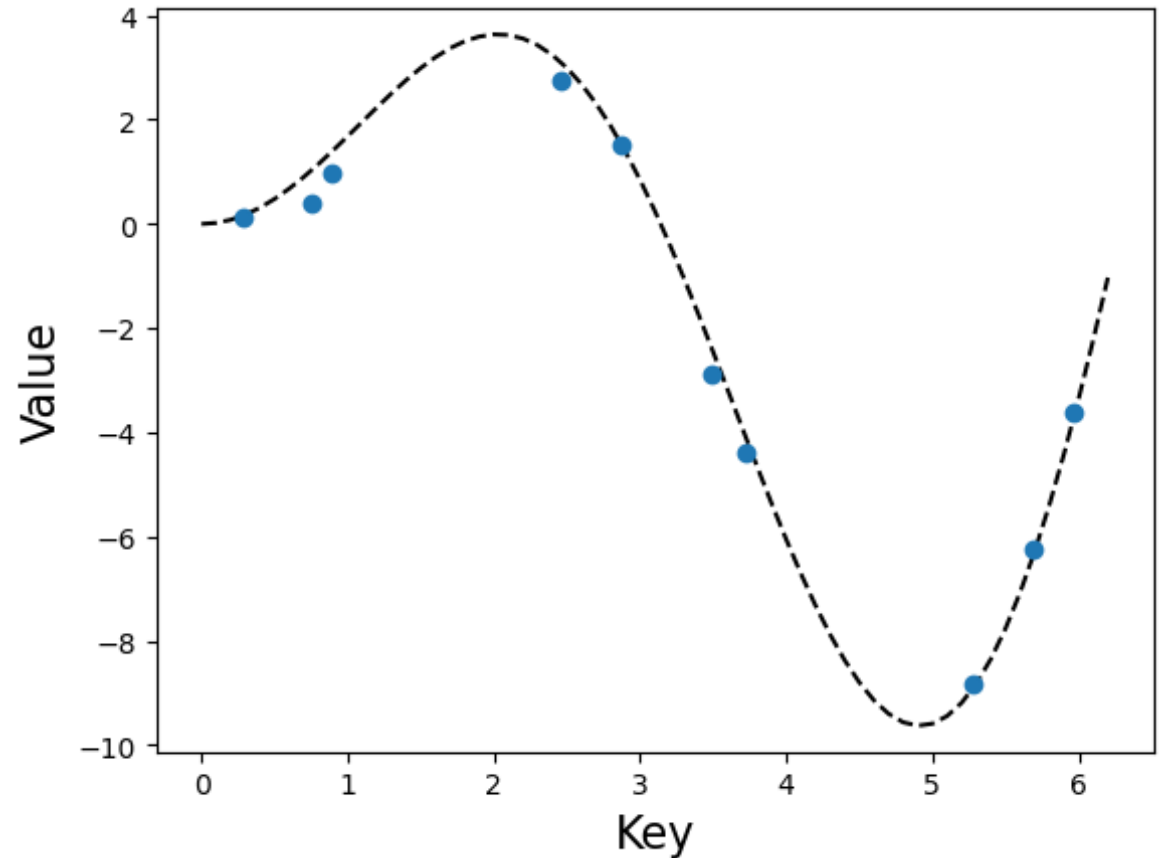
Image taken from: <https://www.librarything.com/mds/>

Attention is... kernel regression

Attention: Analogy & Worked Examples

An old analogy is that of Nadaraya-Watson kernel regression,

1. **Value:** the labels of the training data
2. **Key:** the input features of the training data



Attention is... kernel regression

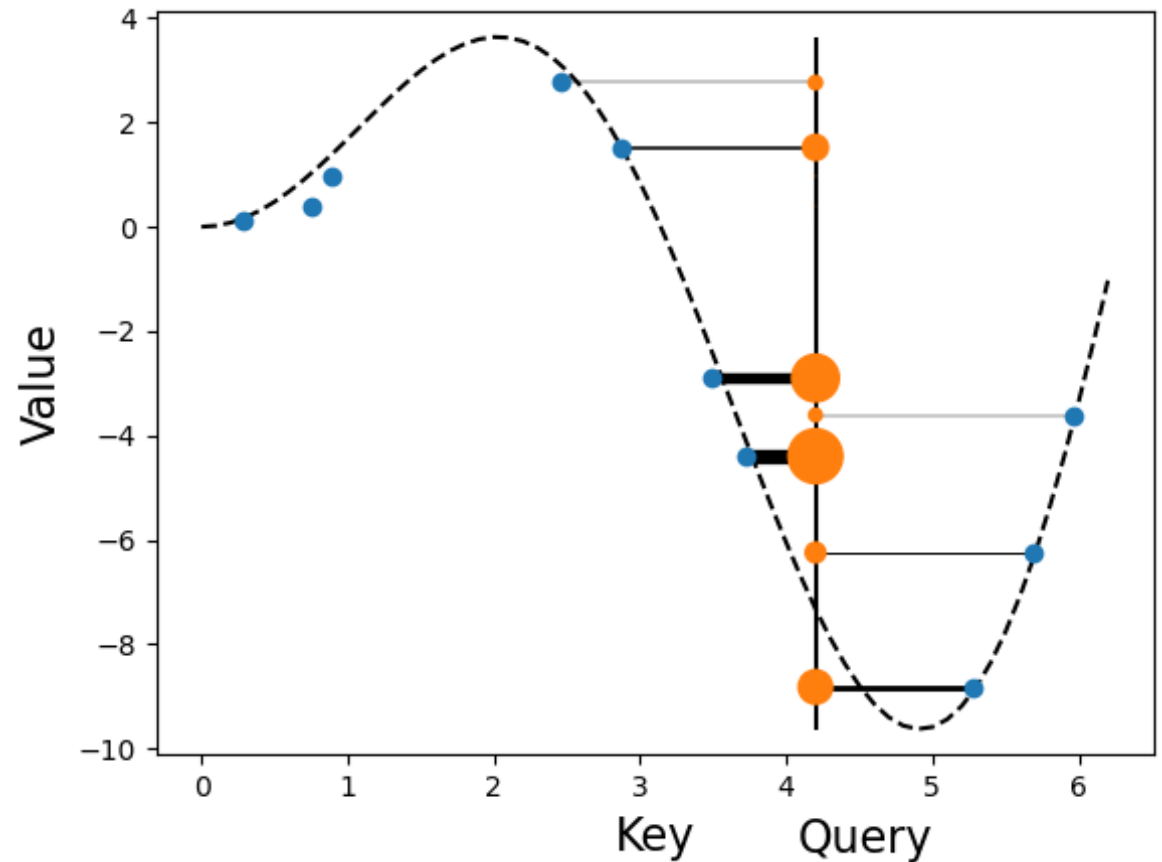
Attention: Analogy & Worked Examples

An old analogy is that of Nadaraya-Watson kernel regression,

1. **Value:** the labels of the training data
2. **Key:** the input features of the training data
3. **Query:** the input feature of the test data
4. **Scoring function:** a heuristic kernel function

Here I use the (log) Gaussian Kernel:

$$\kappa(q, \mathbf{k}) = -\frac{1}{2} \|\mathbf{k} - q\|_2^2$$



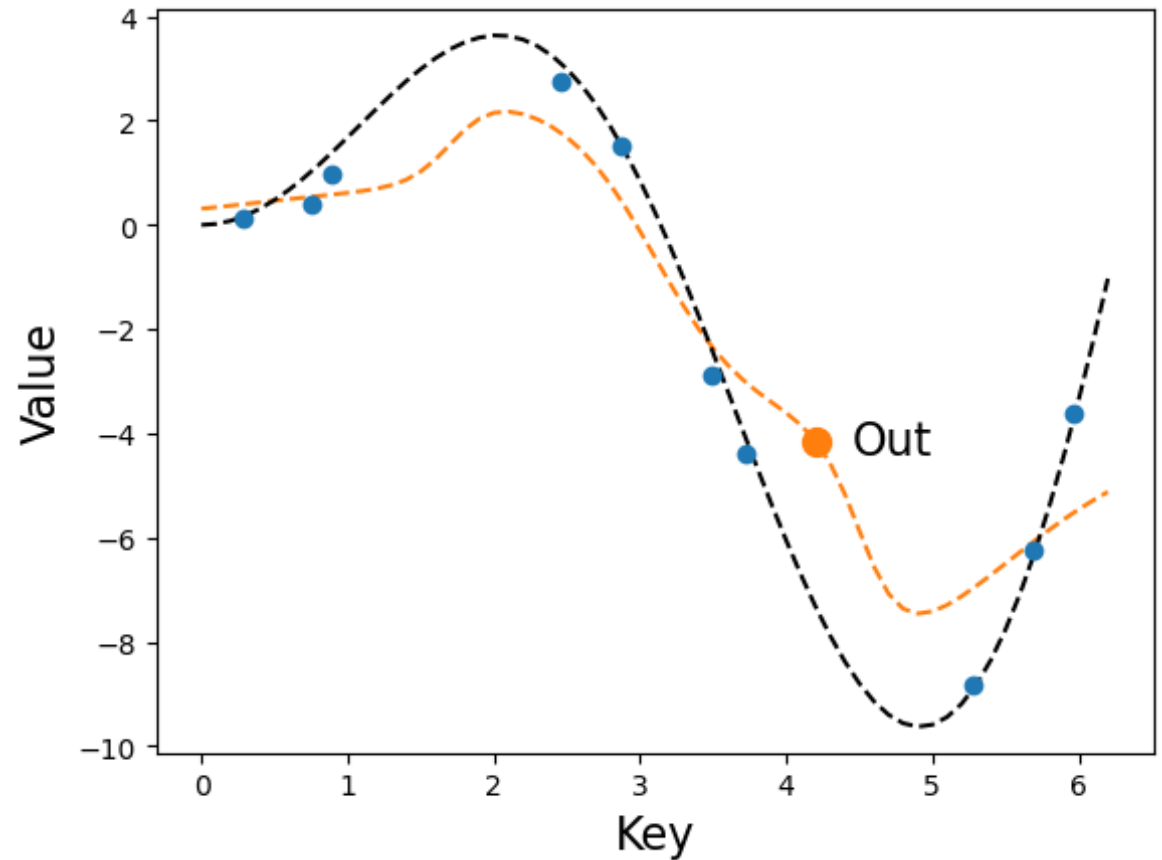
Attention is... kernel regression

Attention: Analogy & Worked Examples

An old analogy is that of Nadaraya-Watson kernel regression,

1. **Value:** the labels of the training data
2. **Key:** the input features of the training data
3. **Query:** the input feature of the test data
4. **Scoring function:** a heuristic kernel function

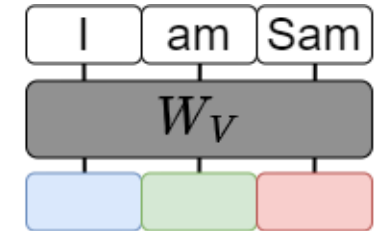
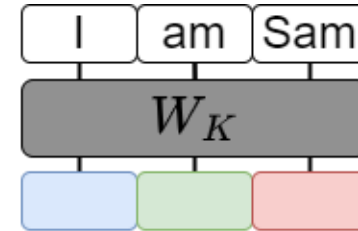
Compute similarity of train features (Key) to test features (Query). The train labels (Value) are weighted by similarity and normalized



Attention is... attention

Attention: Analogy & Worked Examples

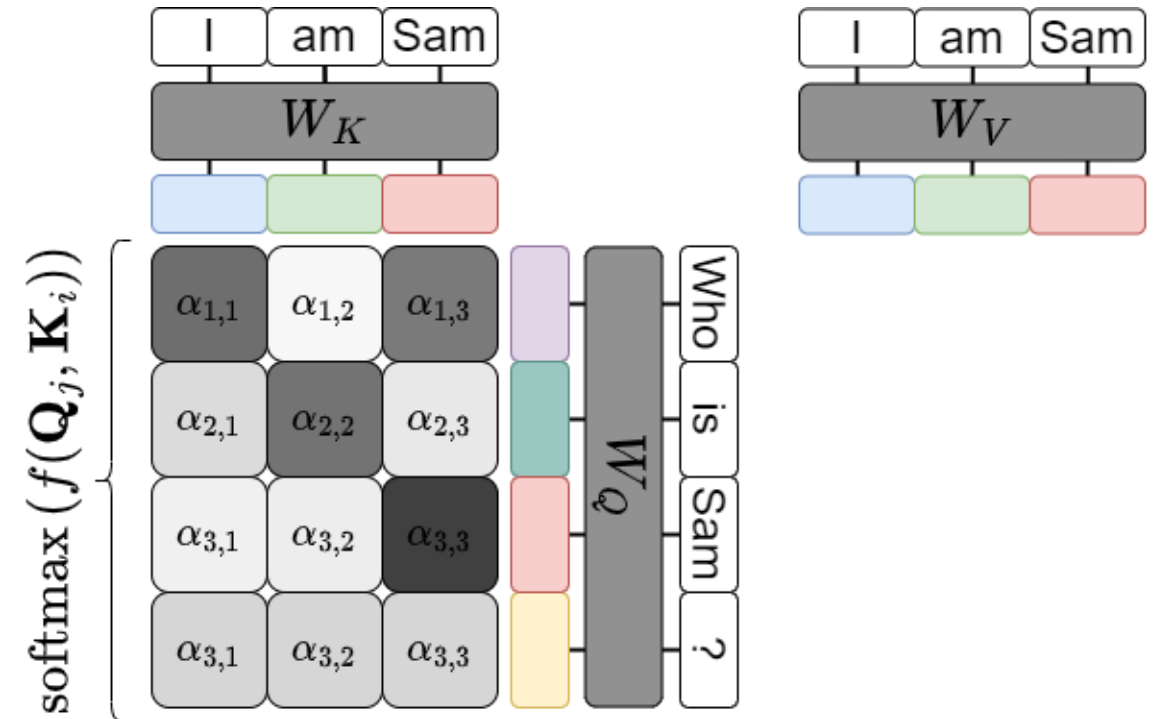
1. **Value:** projected text embeddings
2. **Key:** projected text embeddings



Attention is... attention

Attention: Analogy & Worked Examples

1. **Value:** projected text embeddings
2. **Key:** projected text embeddings
3. **Query:** projected text embeddings (from potentially a different sequence)
4. **Scoring function:** similarity function in embedding space

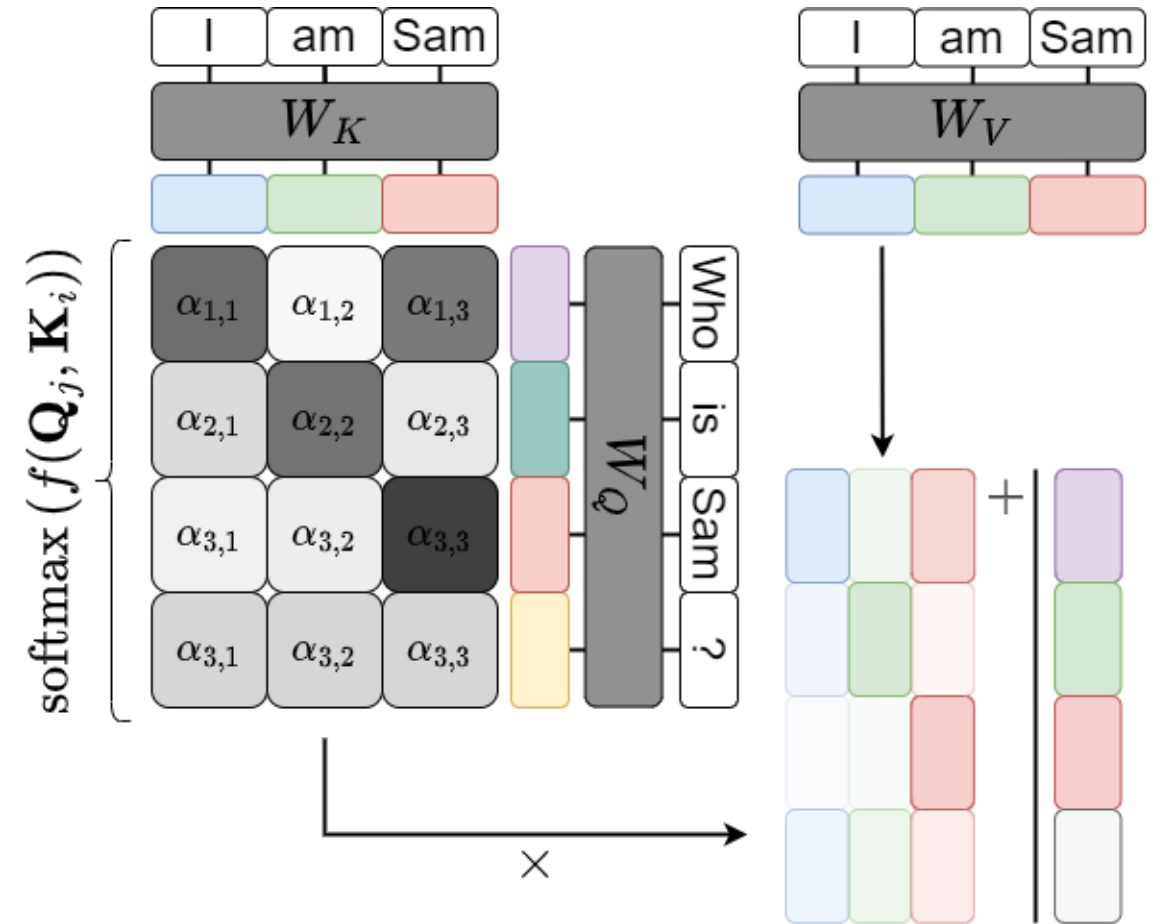


Attention is... attention

Attention: Analogy & Worked Examples

1. **Value:** projected text embeddings
2. **Key:** projected text embeddings
3. **Query:** projected text embeddings (from potentially a different sequence)
4. **Scoring function:** similarity function in embedding space

Compute the similarity of the key and query embeddings. The normalized similarities are used to weight the value embeddings



Attention is a set operation that aggregates sequences based on similarities of embeddings

Examples: Intro

Due to its flexibility, attention is used in many different applications

Attention can aggregate over dimensions other than time

Examples: UDIFY

UDIFY uses attention of layers to summarize a deep model for multi-task learning

1. **Value:** the deep model representations
2. **Key:** N/A
3. **Query:** N/A
4. **Scoring function:** pre-computed

No dynamic weights... does this count?

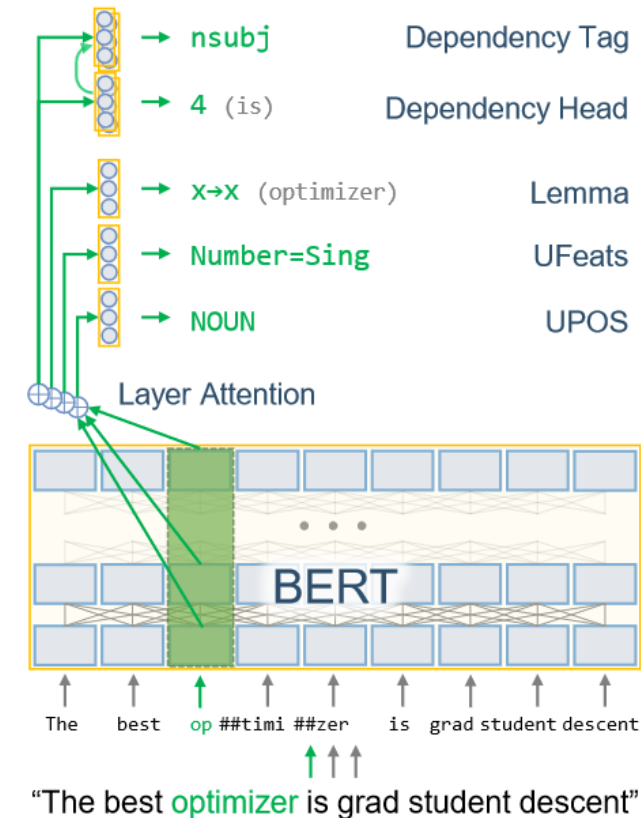


Image taken from: Kondratyuk, D., & Straka, M. (2019). 75 languages, 1 model: Parsing universal dependencies universally. *arXiv preprint arXiv:1904.02099*.

Attention can hierarchically aggregate information at different granularities

Examples: HAN

HAN uses attention to summarize words into sentences, and then sentence into documents

1. **Value:** the representations of the word/sent encoder
2. **Key:** the transformed representations of the word/sent encoder
3. **Query:** static but learned vector
4. **Scoring function:** dot product attention

$$f(\mathbf{u}_i, \mathbf{u}_w) = \mathbf{u}_i^\top \mathbf{u}_w$$

GT: 4 Prediction: 4

pork belly = delicious .
 scallops ?
 i do n't .
 even .
 like .
 scallops , and these were a-m-a-z-i-n-g .
 fun and tasty cocktails .
 next time i 'm in phoenix , i will go
 back here .
 highly recommend .

Image taken from: Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016, June). Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 1480-1489).

Attention across modules allow dynamic processing of long sequences

Examples: Bahdanau et al.

Bahdanau et al. use encoder-decoder attention over words

The encoder processes the source sentence using an RNN

The decoder uses autoregressive RNN generation to produce the next token in the target sentence

But decoder RNN will forget encoder RNN quickly, and global pooling destroys details

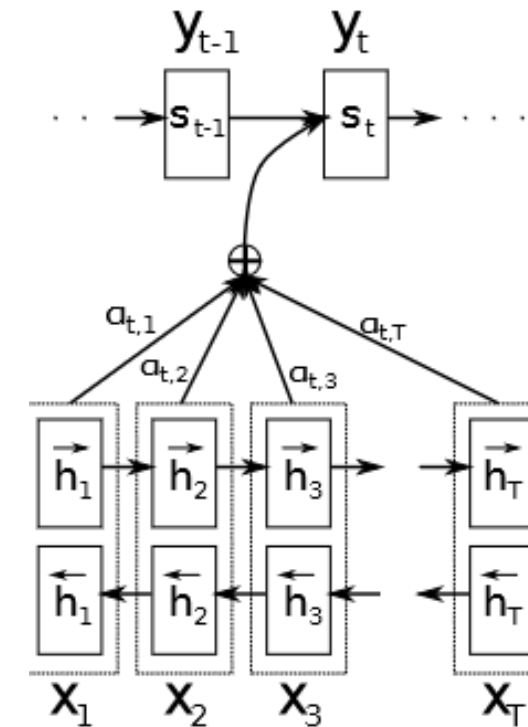


Image taken from: Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Attention across modules allow dynamic processing of long sequences

Examples: Bahdanau et al.

Bahdanau et al. use encoder-decoder attention over words, allowing NMT generation to extend to long sentences

1. **Value:** the representations of the encoder
2. **Key:** the representations of the encoder (?)
3. **Query:** the representations of the decoder (?)
4. **Scoring function:** additive transformation

$$f(s_{i-1}, h_j) = \tanh(s_{i-1} \mathbf{W} + h_j \mathbf{U}_a) \mathbf{V}_a$$

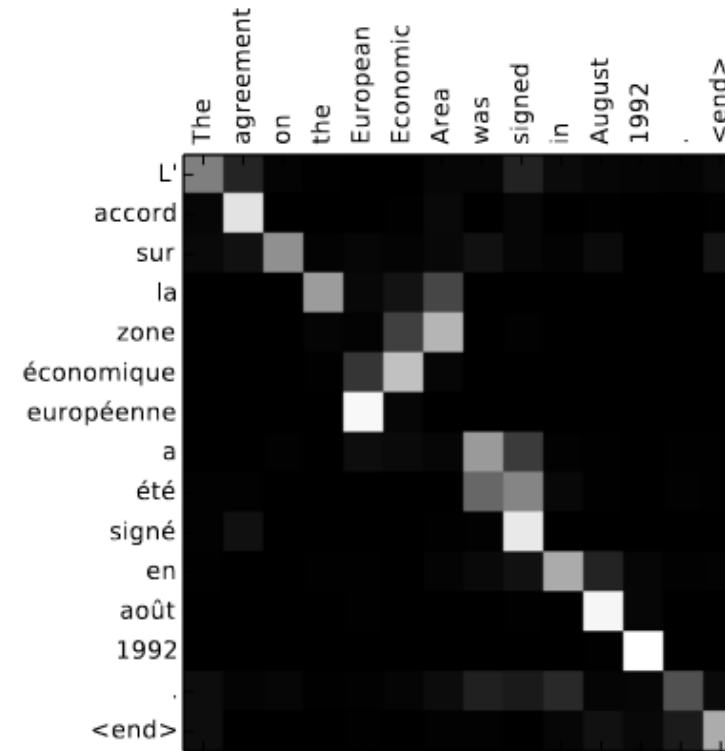


Image taken from: Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Attention makes expressive graph representation learning possible

Examples: GATs

GATv2^[1] uses attention to aggregate a node's neighbourhood for graph learning

1. **Value:** the representation of the node
2. **Key:** the representations of the neighbours
3. **Query:** the representations of the neighbours
4. **Scoring function:** concat transformation

$$f(\mathbf{h}_i, \mathbf{h}_j) = \mathbf{a}^\top \text{LeakyReLU}(\mathbf{W}[\mathbf{h}_i \parallel \mathbf{h}_j])$$

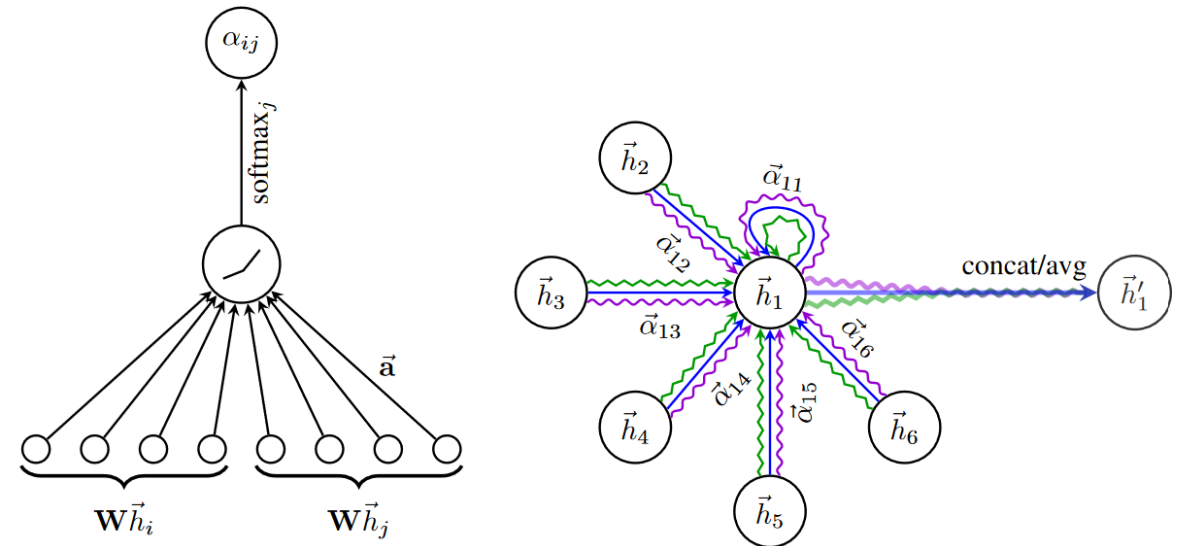


Image taken from: Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

What if... attention is all we need?

Transformers: Introduction

So far, attention is a useful aggregation module,
but not enough on its own

Introducing: the Transformer

106.03762v5 [cs.CL] 6 Dec 2017

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

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin*[‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs: a small fraction of the training costs of the

What if... attention is all we need?

Transformers: Introduction

Massively influential paper, and main topic of today

Graph shows number of papers found when looking up the phrase “pre-trained transformer”

Vaswani et al. (Jun. 2017) sits at 71128 citations

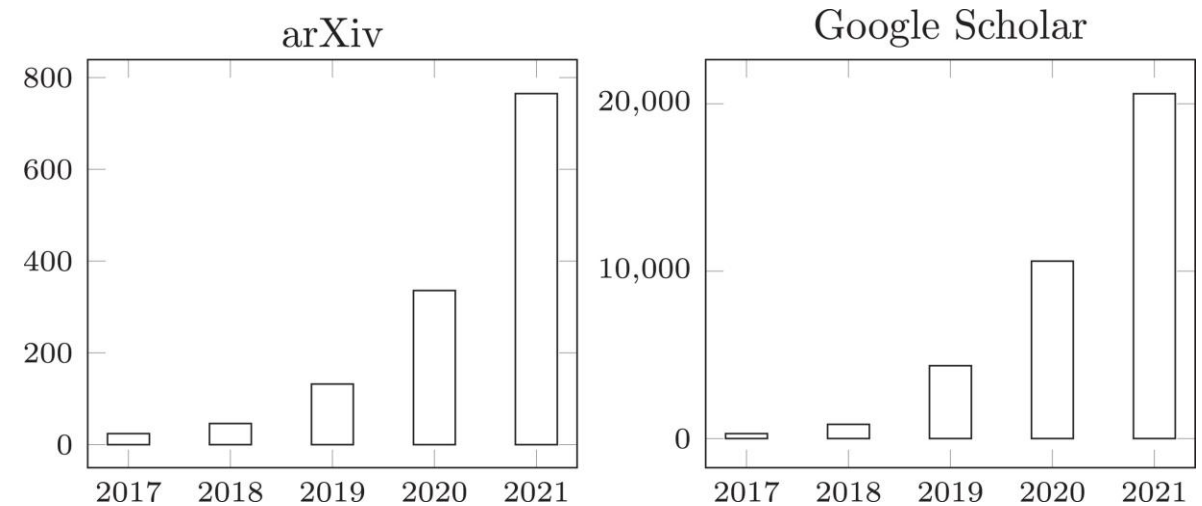


Image taken from: Casola, S., Lauriola, I., & Lavelli, A. (2022). Pre-trained transformers: An empirical comparison. *Machine Learning with Applications*, 9, 100334.

About 33 citations a day...

Once you understand attention, transformers are not too complicated

Transformers: Introduction

Transformers building blocks:

1. Language Modelling head
2. Multi-head scaled dot-product attention mechanisms
 1. Residual connections (Add)
 2. Layer normalization (Norm)
3. Feed forward networks
3. Positional embeddings
4. Word-piece embeddings

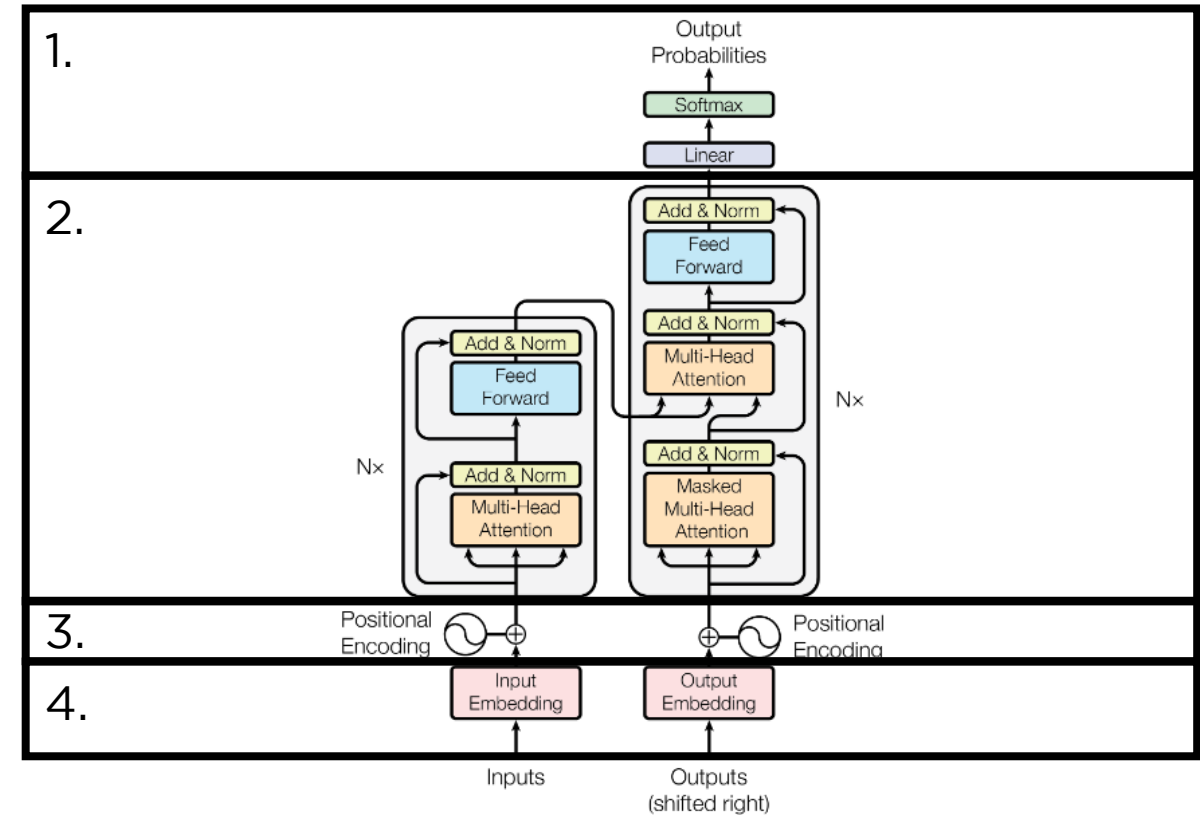


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Once you understand attention, transformers are not too complicated

Transformers: Introduction

Transformers building blocks:

1. ~~Language Modelling head~~
2. Multi-head scaled dot-product attention mechanisms
 1. Residual connections (Add)
 2. Layer normalization (Norm)
 3. Feed forward networks
3. Positional embeddings
4. Word-piece embeddings

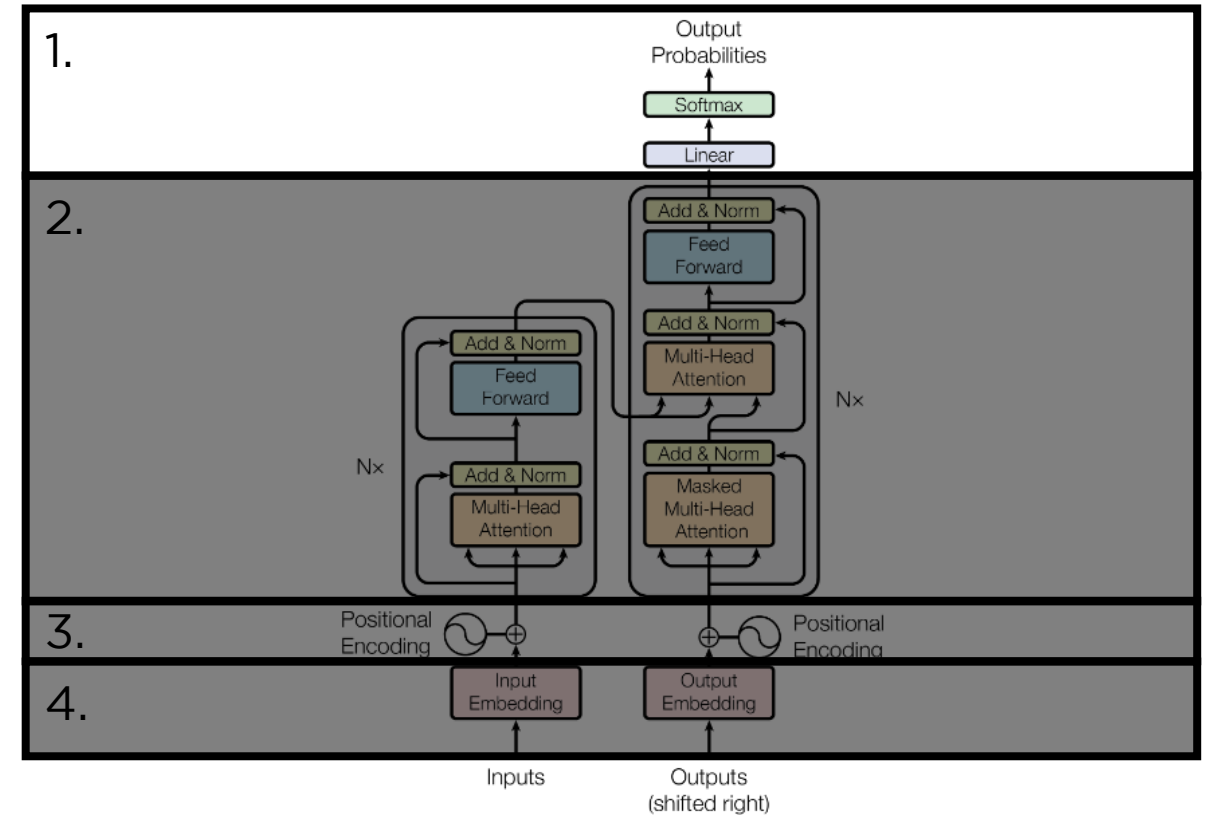


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Once you understand attention, transformers are not too complicated

Transformers: Attention Mechanisms

Transformers building blocks:

1. ~~Language Modelling head~~
2. Multi-head scaled dot-product attention mechanisms
 1. Residual connections (Add)
 2. Layer normalization (Norm)
 3. Feed forward networks
3. Positional embeddings
4. Word-piece embeddings

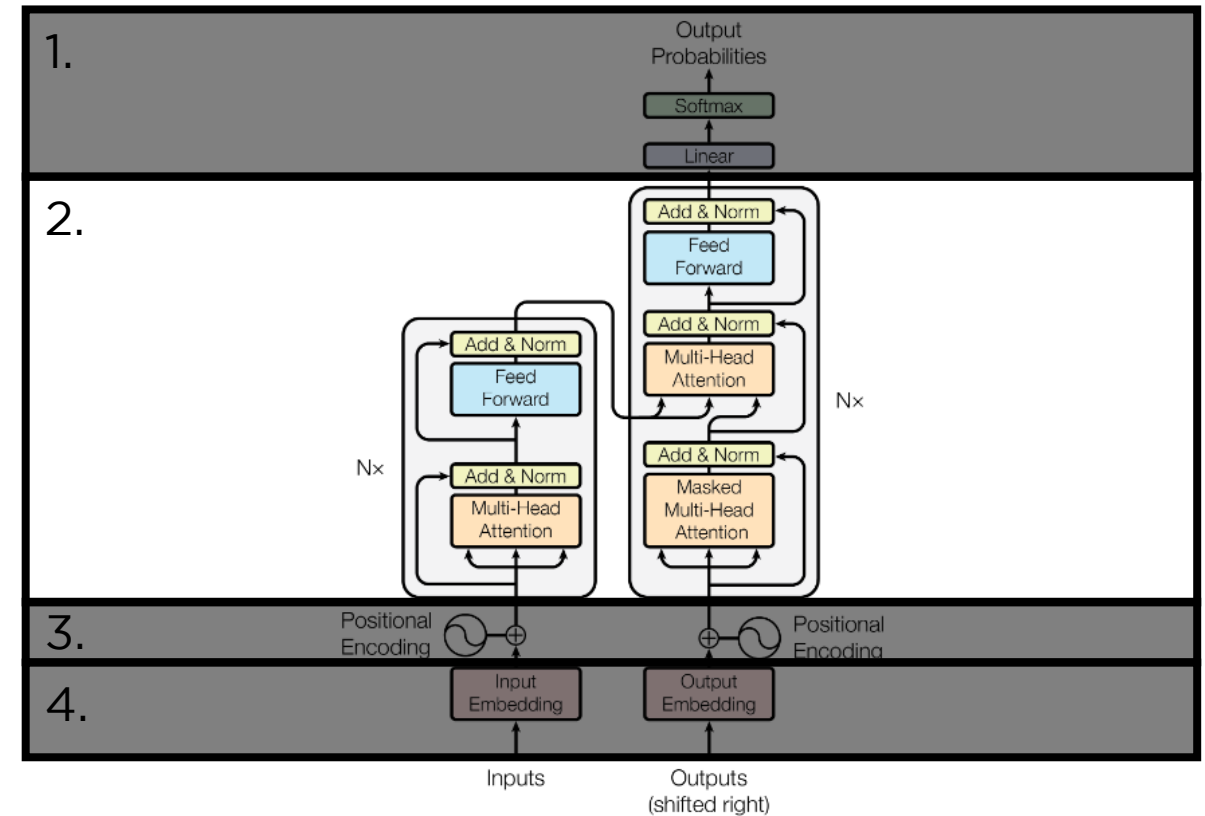


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Transformers use two forms of (self-)attention

Transformers: Attention Mechanisms

Self attention: compute attention from a sequence to itself. Each component gets its own projection. Output is another sequence of the same length

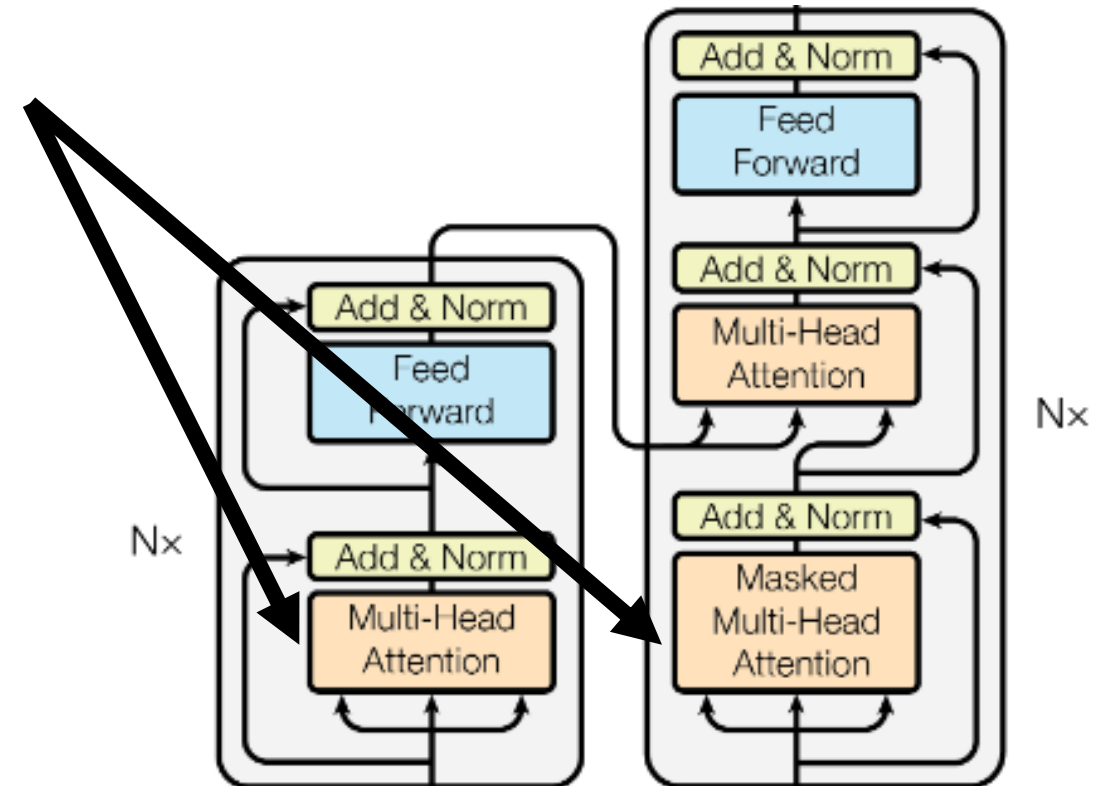


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Transformers use two forms of (self-)attention

Transformers: Attention Mechanisms

Self attention: compute attention from a sequence to itself. Each component gets its own projection. Output is another sequence of the same length

Enc-Dec attention: compute attention from decoder sequence to encoder sequence (encoder provides **V** and **K**, decoder **Q**). Each decoder element can attend to *all* encoder elements

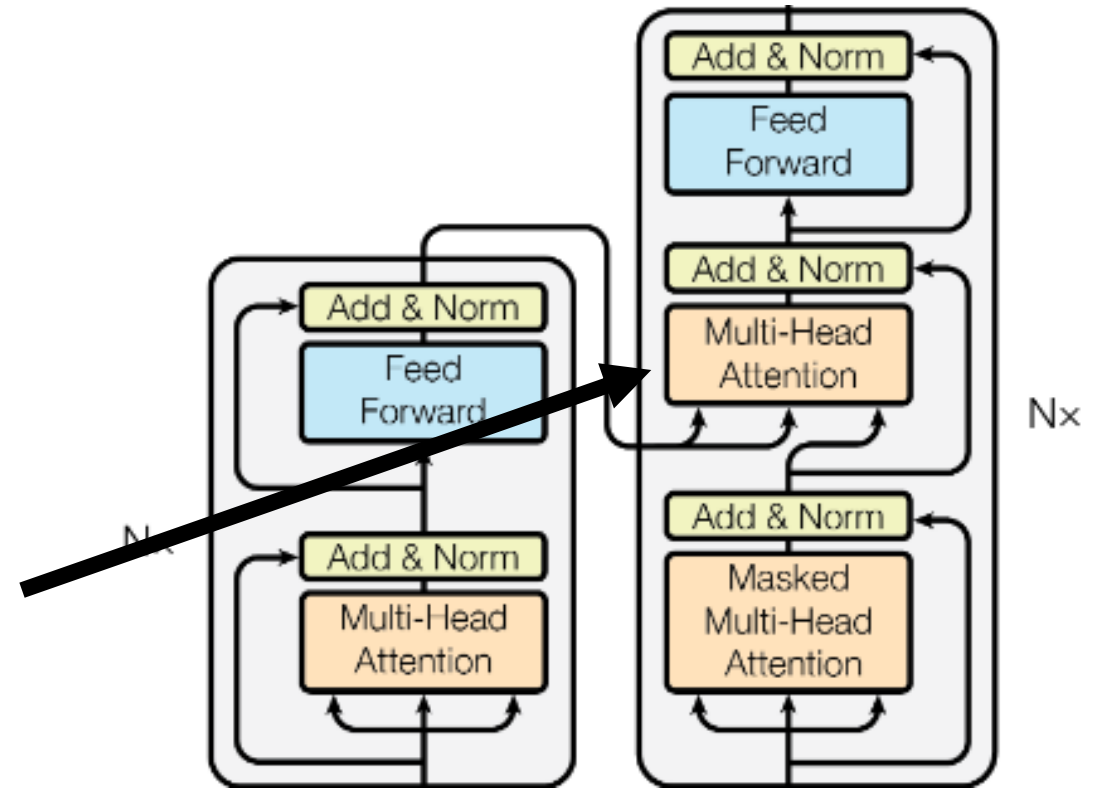


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Transformers use two forms of (self-)attention

Transformers: Attention Mechanisms

Self attention: compute attention from a sequence to itself. Each component gets its own projection. Output is another sequence of the same length

Enc-Dec attention: compute attention from decoder sequence to encoder sequence (encoder provides **V** and **K**, decoder **Q**). Each decoder element can attend to *all* encoder elements

Masking preserves sentence length/causality

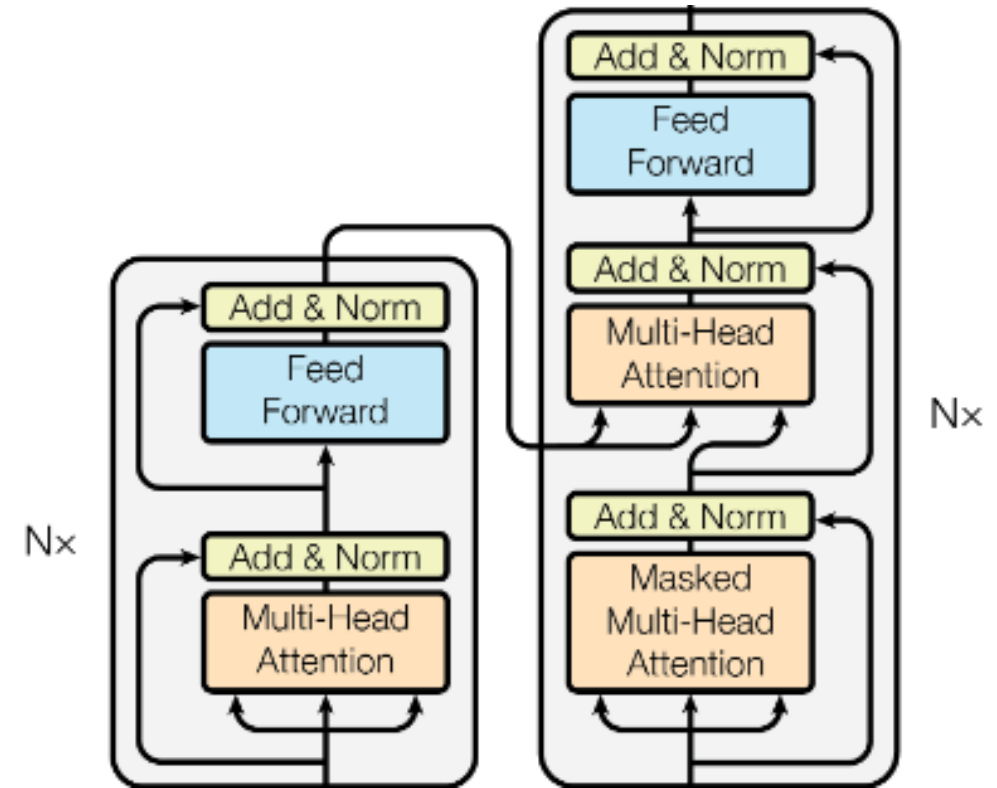


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Self-attention uses scaled dot-products as the scoring function, and all inputs come from the same sequence

Transformers: Scaled Dot-product Self-attention

Scaled-dot product attention follows the recipe

- 1. Value:** a tensor \mathbf{V} [$N \times T_V \times D_V$] of T_V elements, containing content to be aggregated
- 2. Key:** a tensor \mathbf{K} [$N \times T_V \times D_Q$] of T_V elements, containing information *about* content
- 3. Query:** a tensor \mathbf{Q} [$N \times T_Q \times D_Q$] of T_Q elements, containing information about aggregation
- 4. Scoring function:** scaled dot product

$$\text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{D_Q}} \right) \mathbf{V}$$

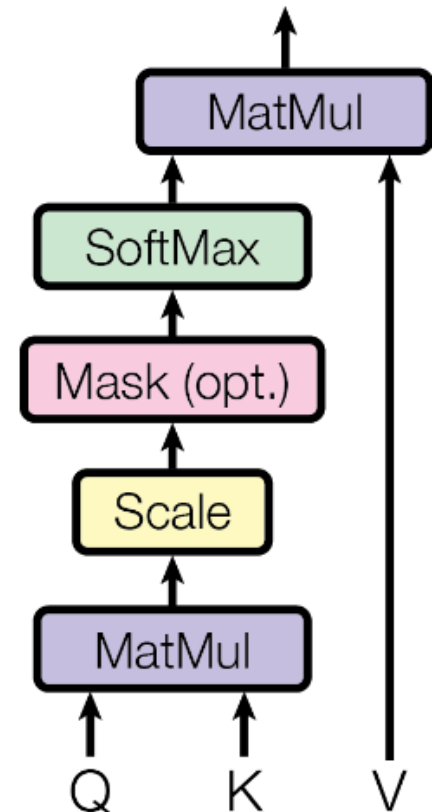


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Masking is necessary to ensure autoregressive behaviour due to lack of recurrence

Transformers: Scaled Dot-product Self-attention

Transformers are designed to avoid recurrence, which is a problem for autoregressive generation

Masking attention ensure tokens only attend to prior tokens

Usually done by setting elements in attention tensor to neginf *before* softmax normalization

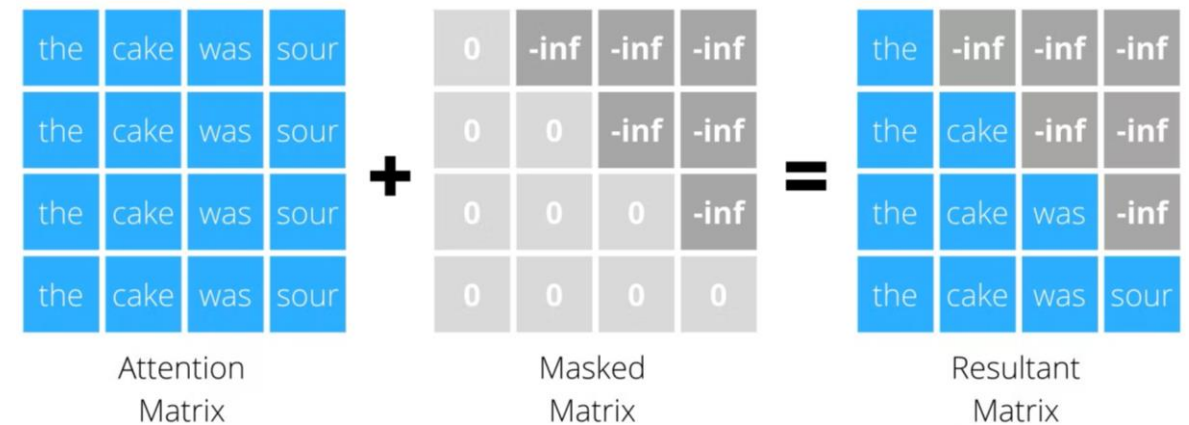


Image taken from: <https://krypticmouse.hashnode.dev/attention-is-all-you-need#heading-finding-positional-encodings>

Self-attention uses scaled dot-products as the scoring function, and all inputs come from the same sequence

Transformers: Scaled Dot-product Self-attention

Self-attention (key, query and value matrices all share the same input **X**) **is not symmetric**

- The contribution of token x_i on x_j is not the same as x_j on x_i

If the projection matrices are shared, self-attention would be symmetric

$$\begin{aligned}\frac{\mathbf{QK}^\top}{\sqrt{D_Q}} &= \frac{\mathbf{XW}_Q (\mathbf{XW}_K)^\top}{\sqrt{D_Q}} \\ &= \frac{\mathbf{XW}_Q \mathbf{W}_K^\top \mathbf{X}^\top}{\sqrt{D_Q}}\end{aligned}$$

Self-attention uses scaled dot-products as the scoring function, and all inputs come from the same sequence

Transformers: Scaled Dot-product Self-attention

Why the scaling?

If $\mathbf{k} \sim \mathcal{N}(0, 1)$ and $\mathbf{q} \sim \mathcal{N}(0, 1)$, i.i.d.

For example, if we're using a batch norm/layer norm

Their dot product is then¹,

$$\mathbf{q}^\top \mathbf{k} \sim \mathcal{N}(0, \sqrt{D_Q})$$

Activation distribution can be normalized again by dividing by standard deviation (the scaling factor)

$$\begin{aligned} \mathbb{E} [\mathbf{q}^\top \mathbf{k}] &= \mathbb{E} \left[\sum_{i=1}^{D_Q} q_i k_i \right] & \text{var} [\mathbf{q}^\top \mathbf{k}] &= \text{var} \left[\sum_{i=1}^{D_Q} q_i k_i \right] \\ &= \sum_{i=1}^{D_Q} \mathbb{E} [q_i k_i] & &= \sum_{i=1}^{D_Q} \text{var} [q_i k_i] \\ &= \sum_{i=1}^{D_Q} \mathbb{E} [q_i] \mathbb{E} [k_i] & &= \sum_{i=1}^{D_Q} \text{var} [q_i] \text{var} [k_i] \\ &= 0 & &= \sum_{i=1}^{D_Q} 1 \\ & & &= D_Q \end{aligned}$$

Self-attention uses scaled dot-products as the scoring function, and all inputs come from the same sequence

Transformers: Scaled Dot-product Self-attention

Their dot product is then¹,

$$\mathbf{q}^\top \mathbf{k} \sim \mathcal{N}(0, \sqrt{D_Q})$$

Recall from DL1, activations that are reasonable bounded help convergence

Even under correlation, assuming $\mathbf{k} \sim \mathcal{N}(0, \sigma)$ and $\mathbf{q} \sim \mathcal{N}(0, \sigma)$, variance should remain bounded^[1]

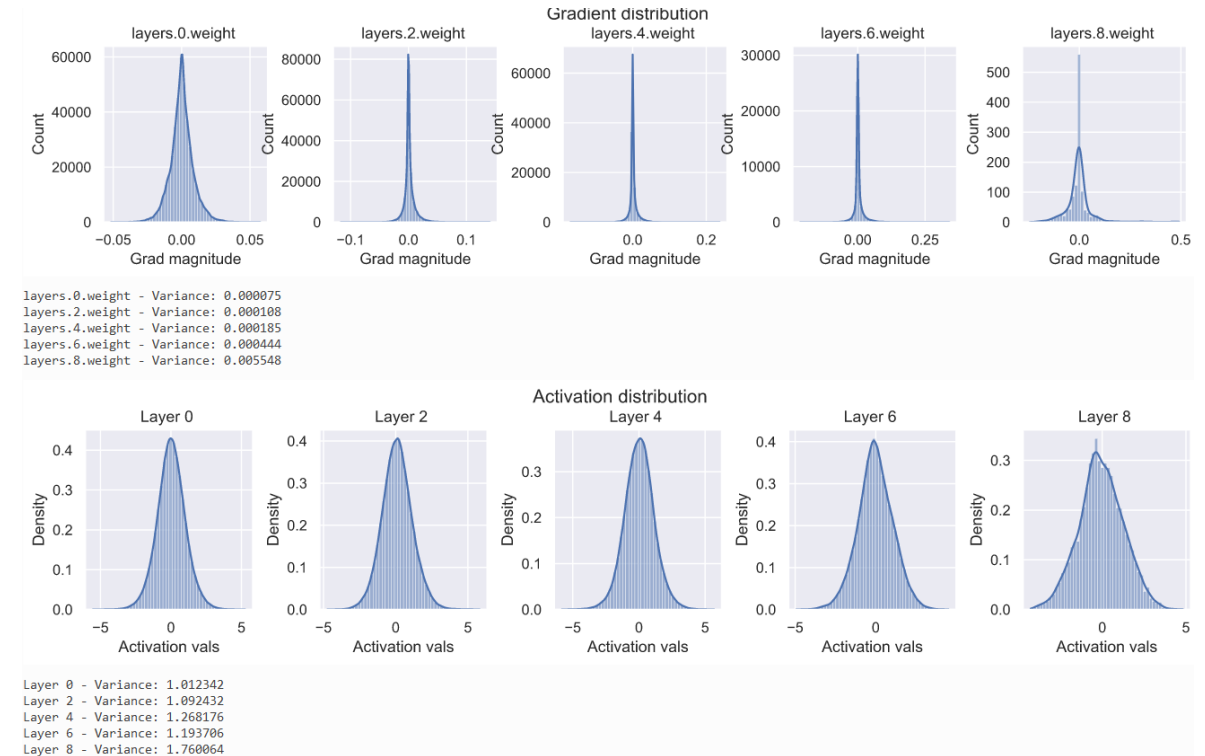


Image taken from: https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial4/Optimization_and_Initialization.html#How-to-find-appropriate-initialization-values

Multiple heads allow attending multiple distinct concepts in parallel

Transformers: Multi-headed Self-attention

Multi-headed attention increases flexibility with minimal computational overhead

For h heads, reduce D_V to $\frac{D_V}{h}$, run each head in parallel, then concatenate the output together ($h \times \left(\frac{D_V}{h}\right) = D_V$). Linearly transform output again to mix heads into single output

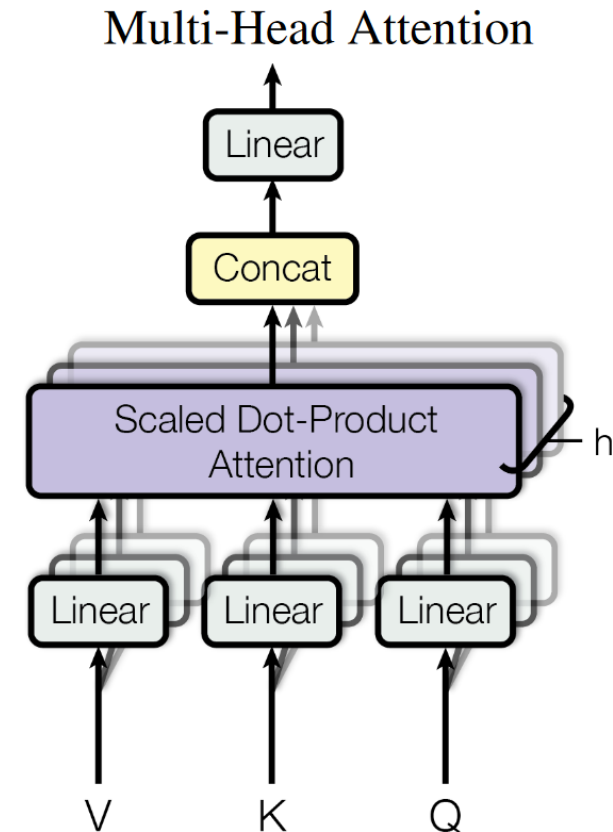


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Multiple heads allow attending multiple distinct concepts in parallel

Transformers: Multi-headed Self-attention

Why are multiple heads needed?

“Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. [One] attention head, averaging inhibits this.”

Different heads produce different attention patterns (and have different tasks)

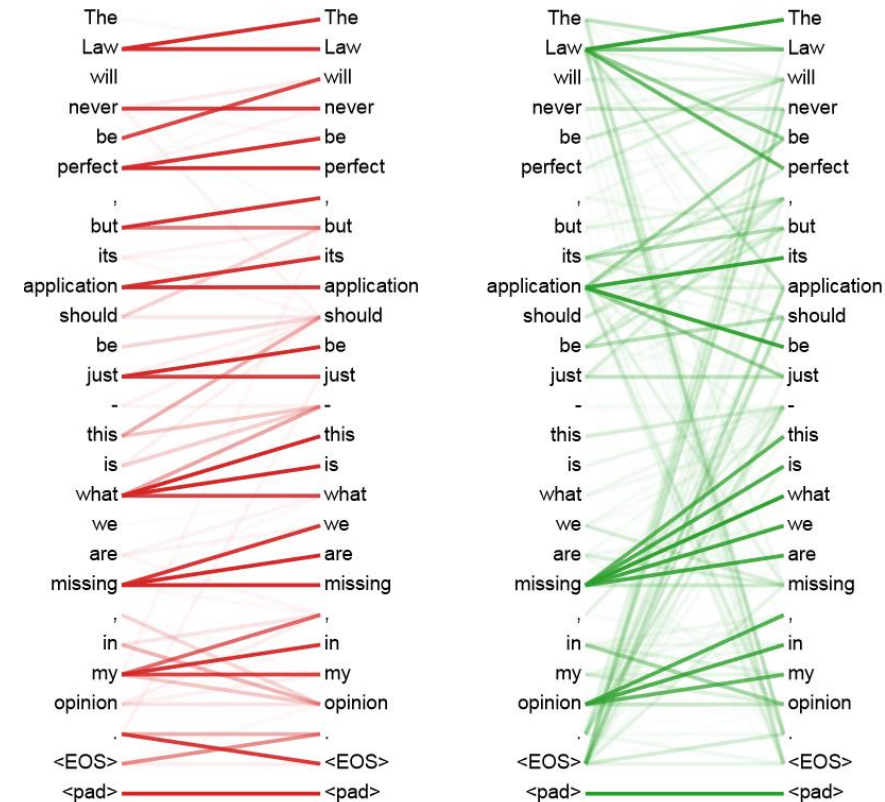


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Multiple heads allow attending multiple distinct concepts in parallel

Transformers: Multi-headed Self-attention

But... [1]

“Even though, by themselves, heads are not low rank, the product of their concatenation K is low rank. Hence, the heads are sharing common projections in their column-space.”

Then again... [1]

“... encoder-decoder attention is much more dependent on multi-headedness than self-attention.”

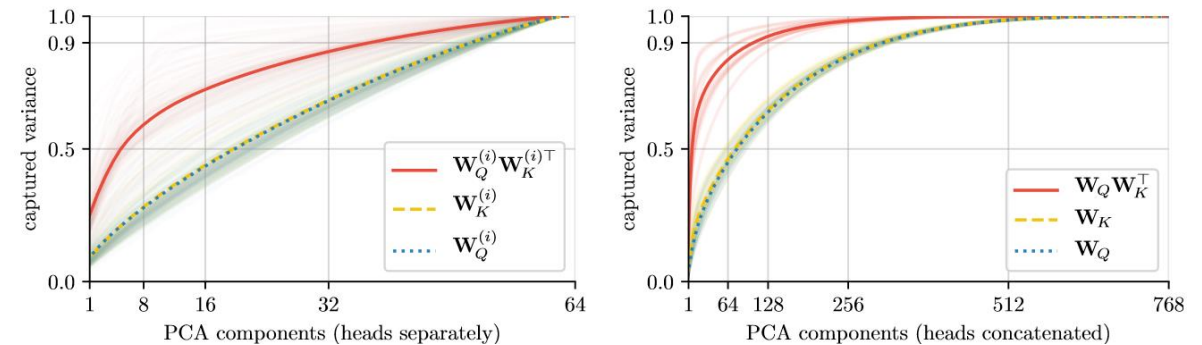


Image taken from: Cordonnier, J. B., Loukas, A., & Jaggi, M. (2020). Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*.

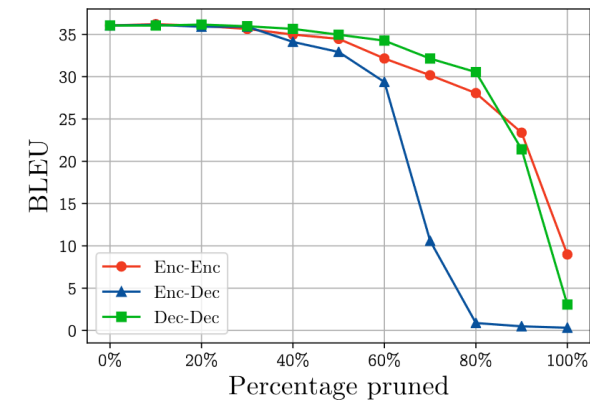


Image taken from: Michel, P., Levy, O., & Neubig, G. (2019). Are sixteen heads really better than one?. *Advances in neural information processing systems*, 32.

Transformers encode first, then decode in autoregressive manner

Transformers: High-level Information Flow

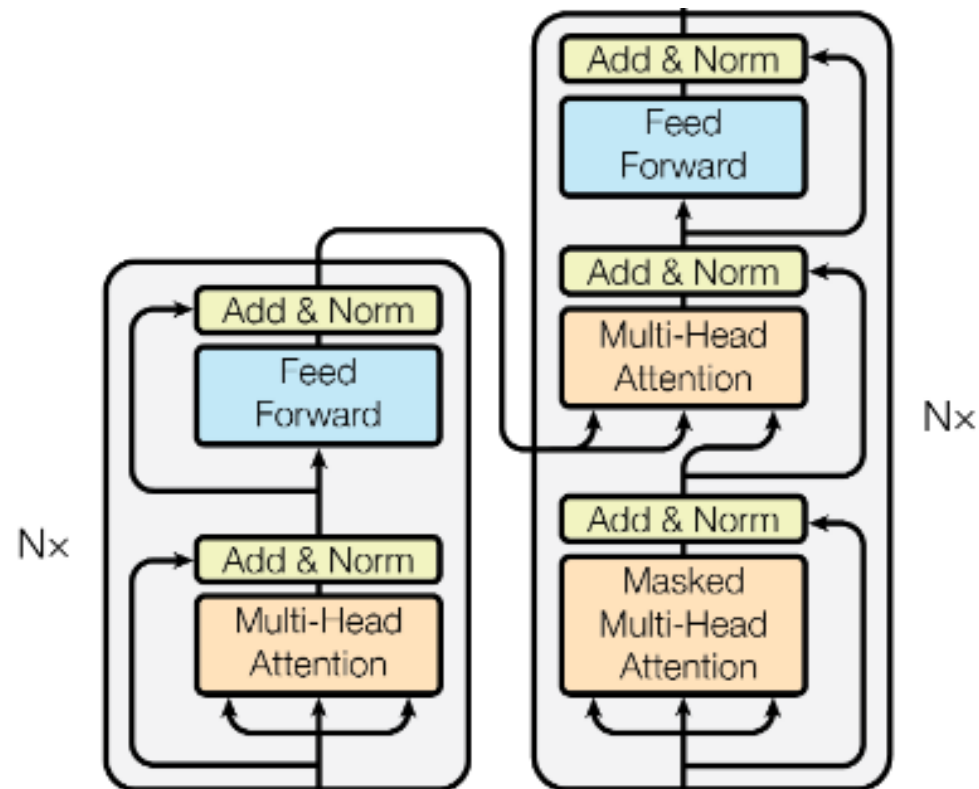


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Image taken from: <https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

Transformer blocks consist of more than only attention

Transformers: Non-attention Modules

Transformers building blocks:

1. ~~Language Modelling head~~
2. ~~Multi-head scaled dot-product attention mechanisms~~
 1. Residual connections (Add)
 2. Layer normalization (Norm)
 3. Feed forward networks
3. Positional embeddings
4. Word-piece embeddings

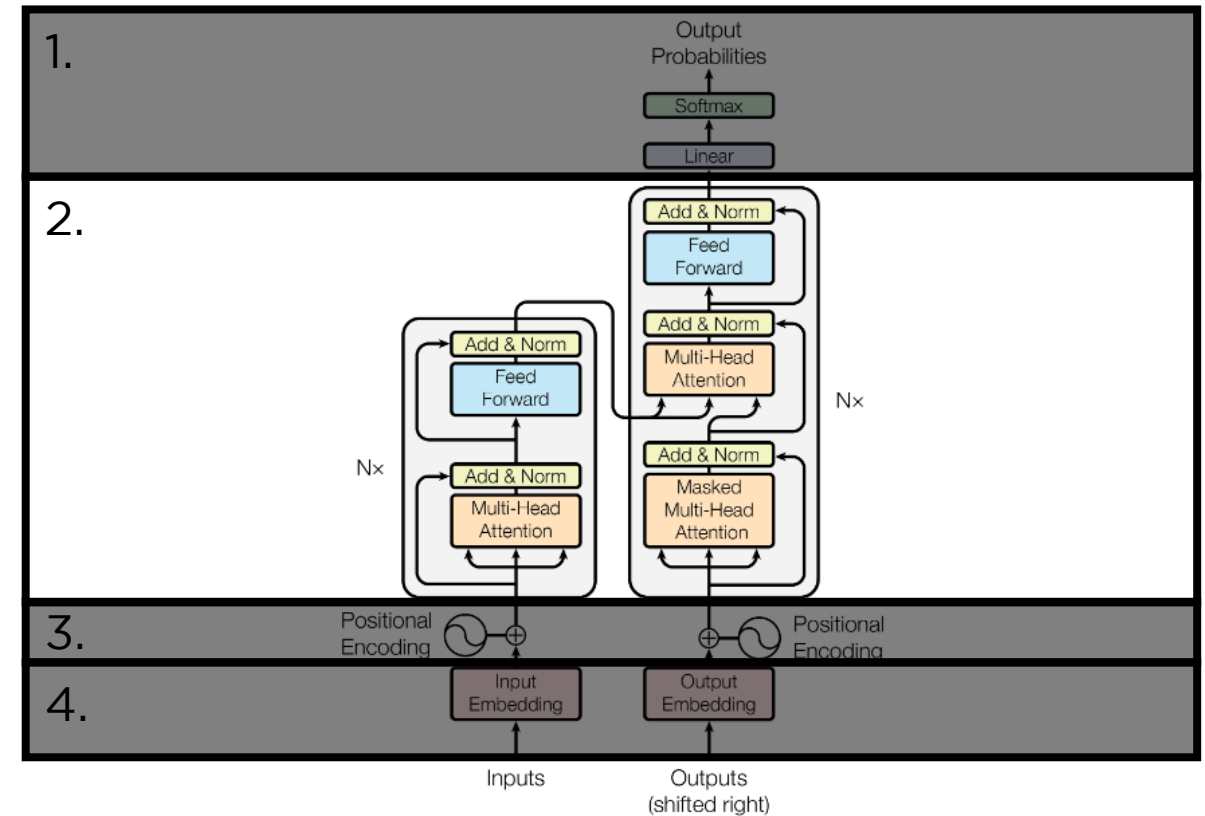


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Residual connections and layer normalization speed up convergence, and propagate lower level information

Transformers: Non-attention Modules

After each module/‘sub-layer’ previous input is added to output, and normalized together

Paper does not specify why. Default reasoning:

1. Speeds up convergence
2. Stabilizes training
3. Retains embedding information

$$\mathbf{X}_l = \text{LayerNorm}(\mathbf{X}_{l-1} + \text{SubLayer}(\mathbf{X}_{l-1}))$$

Tranformers are deep (GPT-3 has 12-96 layers)
so residual connections are a must

Residual connections and layer normalization speed up convergence, and propagate lower level information

Transformers: Non-attention Modules

Why LayerNorm?¹

1. Convention in NLP (from RNN days)
2. LayerNorm is independent of sequence length
3. Text shows high variance in feature statistics across batches

$$\mathbf{X}_l = \text{LayerNorm}(\mathbf{X}_{l-1} + \text{SubLayer}(\mathbf{X}_{l-1}))$$

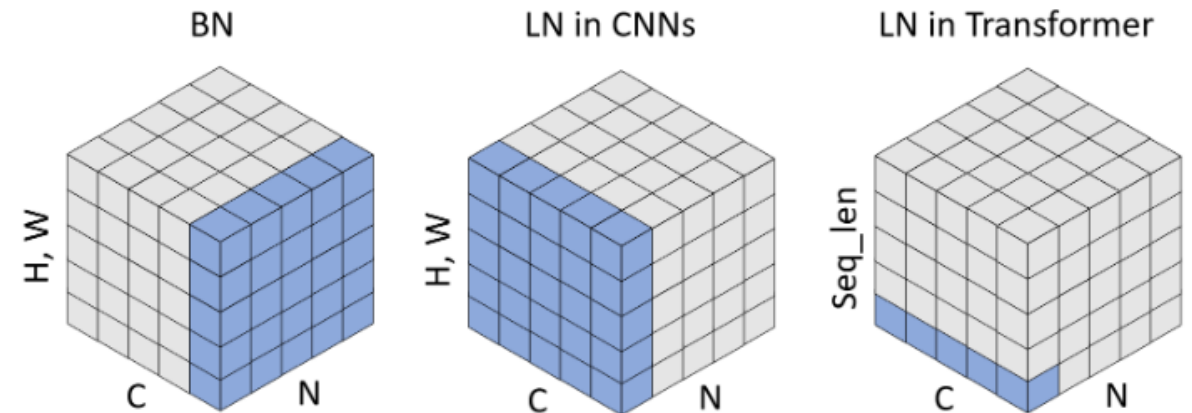


Image taken from: Yao, Z., Cao, Y., Lin, Y., Liu, Z., Zhang, Z., & Hu, H. (2021). Leveraging batch normalization for vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 413-422).

Wide FFN add non-linearity in a parallelizable manner

Transformers: Non-attention Modules

After attention, feed $\mathbf{X} [N \times T \times D_{model}]$ through a *wide* 2 layer feed-forward network

1. First project to D_{ff} (e.g. 2048), apply non-linearity
2. Then project back to D_{model} (e.g. 512) and output

$$\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2$$

Applied in parallel to every token in batch

Wide preferred over deep as it increases the parallelization (at the cost of some flexibility)

Attention is not all you need...

Transformers: Non-attention Modules

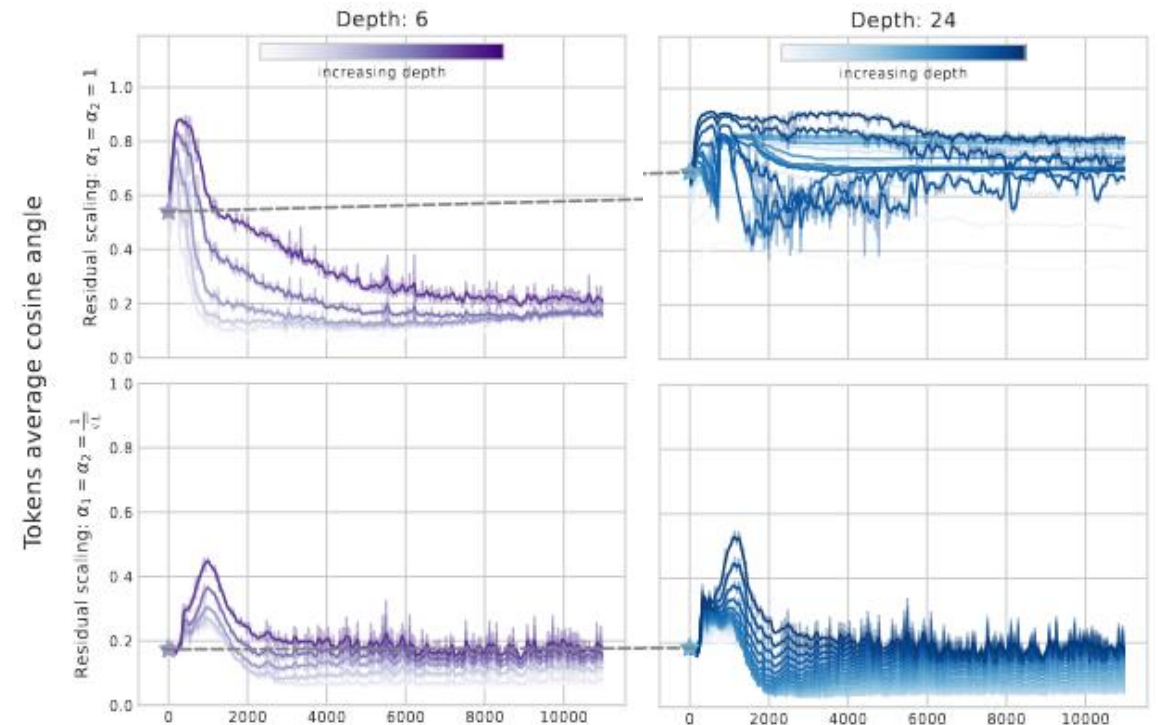
Dong et al.¹ prove transformers with *only* self-attention decay to “token uniformity” **as depth increases**, double exponentially (very fast)

To avoid this, in order of importance:

1. Residual connections
2. Feed-forward network

For this particular problem, LayerNorm has no effect

Related to oversmoothing in GNNs?



Taken from: Noci, L., Anagnostidis, S., Biggio, L., Orvieto, A., Singh, S. P., & Lucchi, A. (2022). Signal Propagation in Transformers: Theoretical Perspectives and the Role of Rank Collapse. *arXiv preprint arXiv:2206.03126*.

Position encoding add necessary information about sequence position of tokens

Transformers: Position Encoding

Transformers building blocks:

1. ~~Language Modelling head~~
2. ~~Multi-head scaled dot-product attention mechanisms~~
 1. ~~Residual connections (Add)~~
 2. ~~Layer normalization (Norm)~~
 3. ~~Feed forward networks~~
3. Positional embeddings
4. Word-piece embeddings

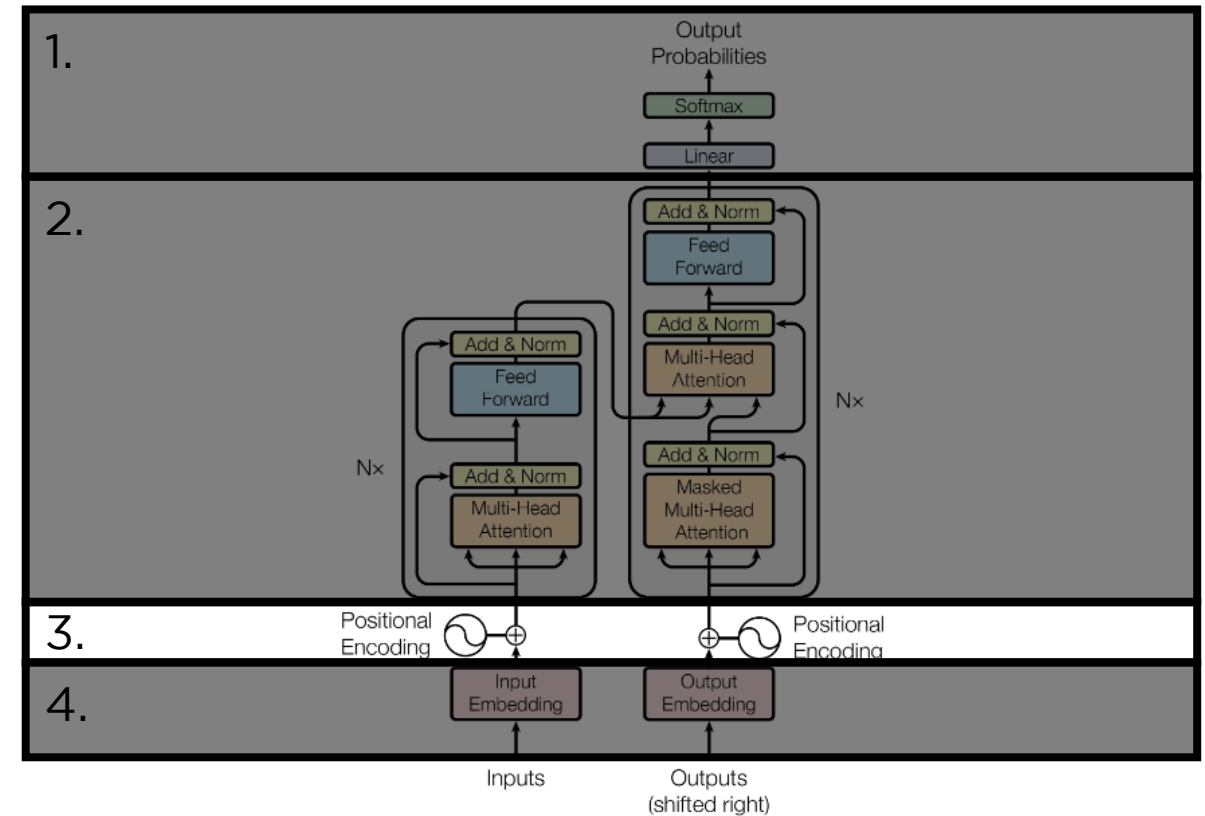


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Position encoding add necessary information about sequence position of tokens

Transformers: Position Encoding

We want to encode position using some low-cost perturbation. Requirements are:

1. Deterministic
2. Distinct at all time-steps
3. Easily extends to long sequences
4. Also encodes relative positions well

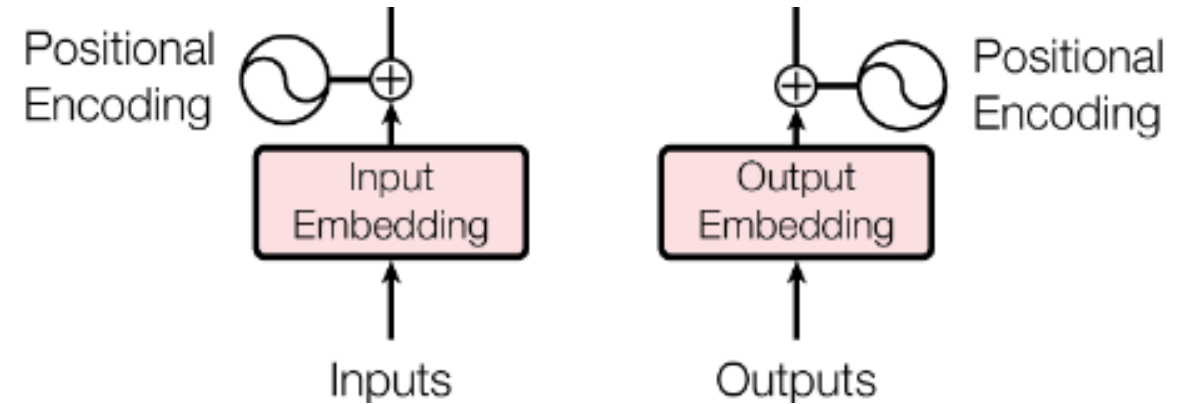


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Positions must be uniquely identifiable for all potential sequence lengths

Transformers: Position Encoding

Proposal 1: append the bit representation of the sequence position. Left bit oscillates slowly, right bit quickly

1. Deterministic [✓]
2. Distinct at all time-steps [✓]
3. Easily extends to long sequences [~]
4. Also encodes relative positions well [✓]

... but we want to stay with floating point numbers, and keep sequence length

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

Image taken from: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Sinusoids of varying frequency serve as a continuous alternative of appending bits

Transformers: Position Encoding

Proposal 2: use sinusoids of varying frequency as continuous relaxation of bits

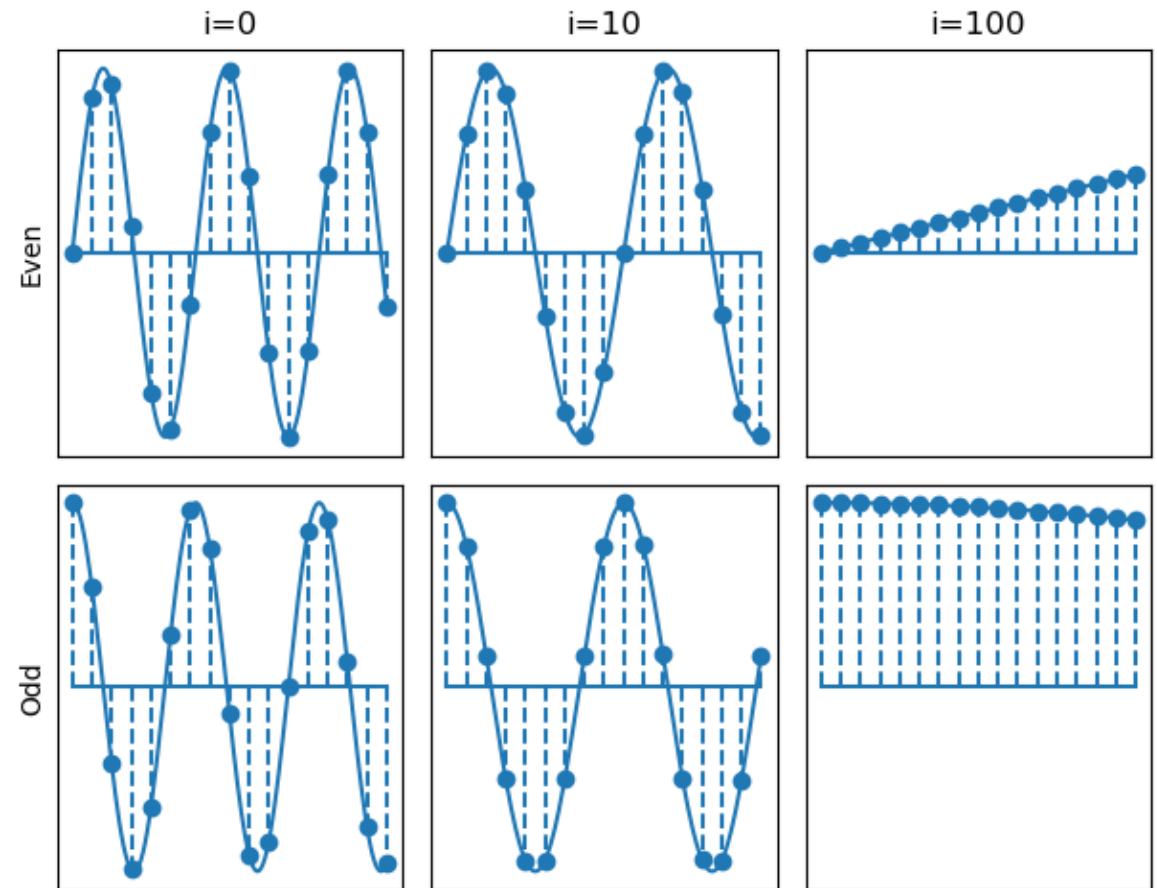
$$\text{PE}_{t,i}^{(\text{even})} = \sin \left(\frac{1}{k^{2i/D_Q}} \cdot t \right)$$

$$\text{PE}_{t,i}^{(\text{odd})} = \cos \left(\frac{1}{k^{2i/D_Q}} \cdot t \right)$$

Even vector positions are sines, odd are cosines

Frequency **decreases** with vector position (k is some very large number, e.g. 10000)

Sample at discrete time-steps



Sinusoids of varying frequency serve as a continuous alternative of appending bits

Transformers: Position Encoding

Proposal 2: use sinusoids of varying frequency as continuous relaxation of bits

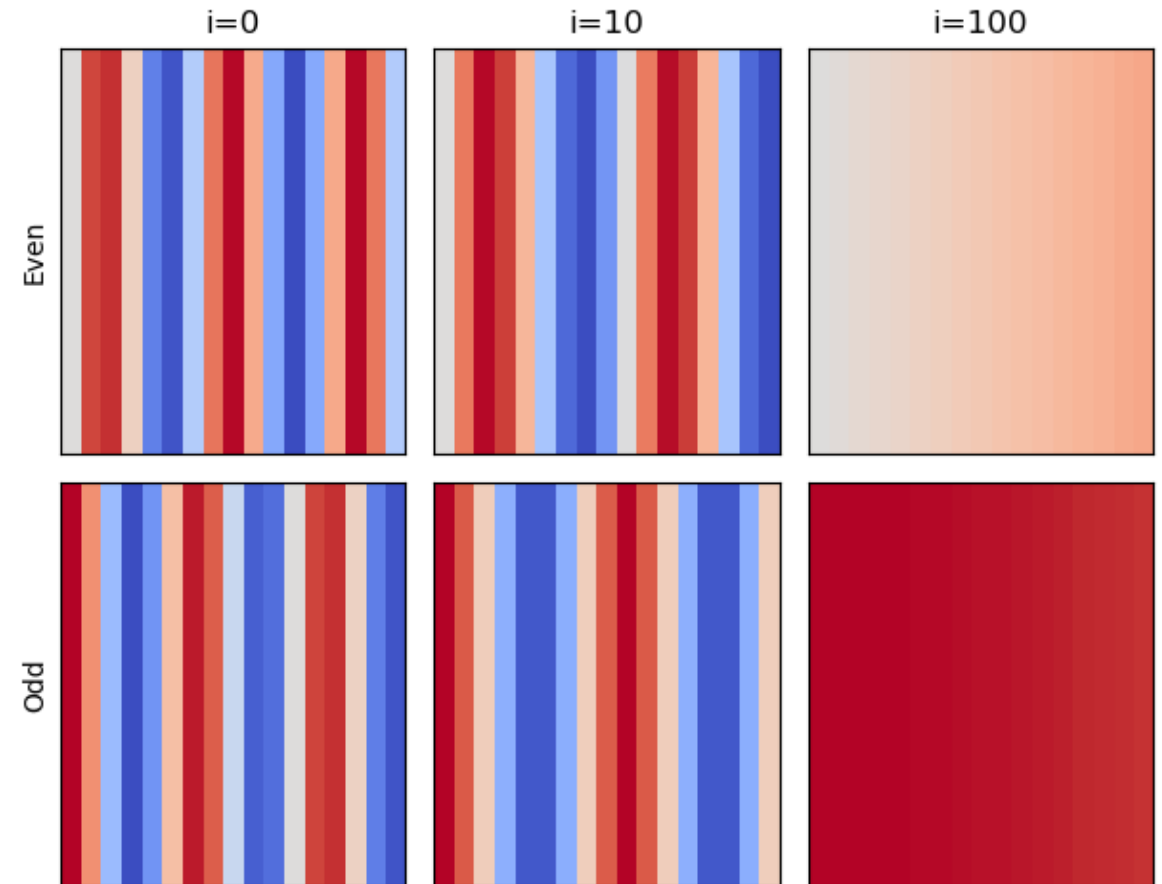
$$\text{PE}_{t,i}^{(\text{even})} = \sin \left(\frac{1}{k^{2i/D_Q}} \cdot t \right)$$

$$\text{PE}_{t,i}^{(\text{odd})} = \cos \left(\frac{1}{k^{2i/D_Q}} \cdot t \right)$$

Even vector positions are sines, odd are cosines

Frequency **decreases** with vector position (k is some very large number, e.g. 10000)

Sample at discrete time-steps



Sines and cosines alternate positions in the position embedding matrix, creating a complex pattern

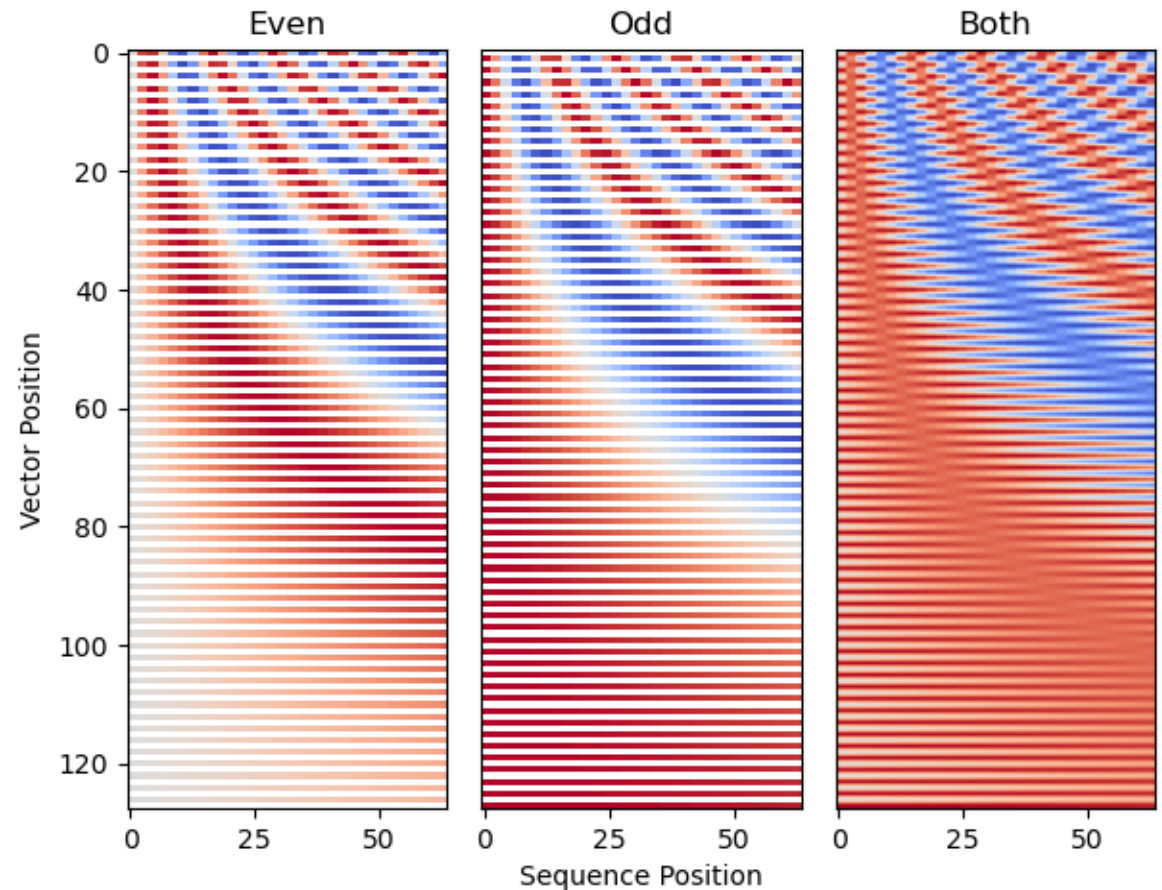
Transformers: Position Encoding

Proposal 2: use sinusoids of varying frequency as continuous relaxation of bits

Interleave the even and odd positions

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}$$

Image taken from:
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/



Alternating sines and cosines create a notion of relative embeddings, making learning easier

Transformers: Position Encoding

Proposal 2: use sinusoids of varying frequency as continuous relaxation of bits

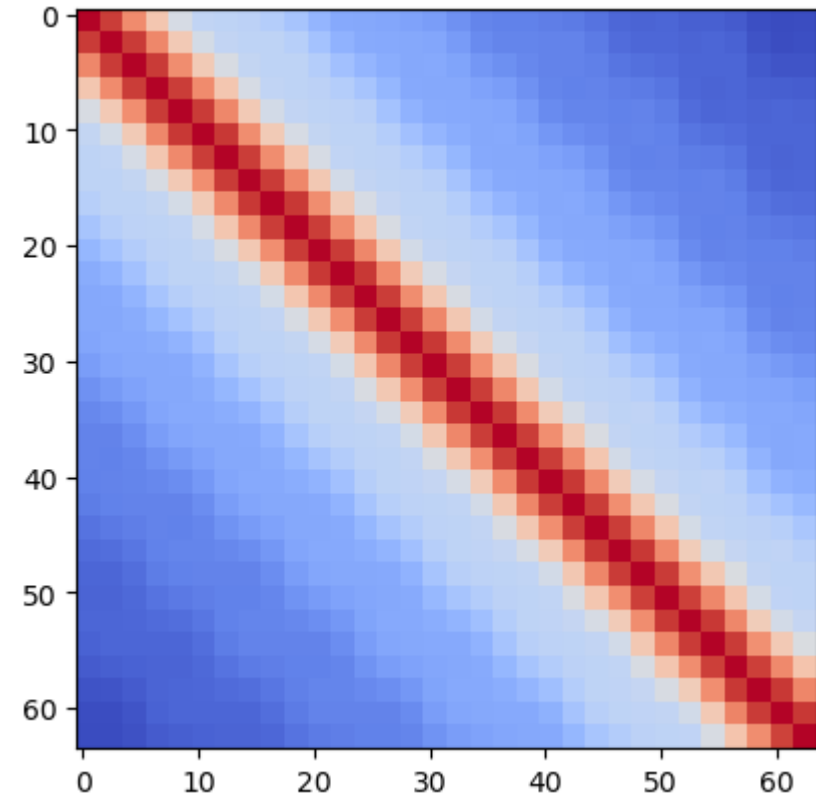
Relative positions are:

1. A linear transformation of each other, dependent only on relative position

$$A \begin{pmatrix} \sin(\omega_i \cdot t) \\ \cos(\omega_i \cdot t) \end{pmatrix} = \begin{pmatrix} \sin(\omega_i \cdot t + \psi) \\ \cos(\omega_i \cdot t + \psi) \end{pmatrix}$$

2. Symmetric and decaying with distance

“... we hypothesized it would allow the model to easily learn to attend by relative positions”^[1]



Learned embeddings yield identical results, but with theoretical drawbacks

Transformers: Position Encoding

Proposal 3: learn the offset using a simple embedding lookup module

1. Deterministic [✓]
2. Distinct at all time-steps [?]
3. Easily extends to long sequences [x]
4. Also encodes relative positions well [?]

Limits maximum encoding size, but works equally well (for original transformer)

	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
(E)	positional embedding instead of sinusoids									4.92	25.7	

Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

To constrain parameter count and extend to unseen word-forms, Transformers use sub-word tokens

Transformers: Tokenization

Transformers building blocks:

1. ~~Language Modelling head~~
2. ~~Multi-head scaled dot-product attention mechanisms~~
 1. ~~Residual connections (Add)~~
 2. ~~Layer normalization (Norm)~~
 3. ~~Feed forward networks~~
3. ~~Positional embeddings~~
4. Word-piece embeddings

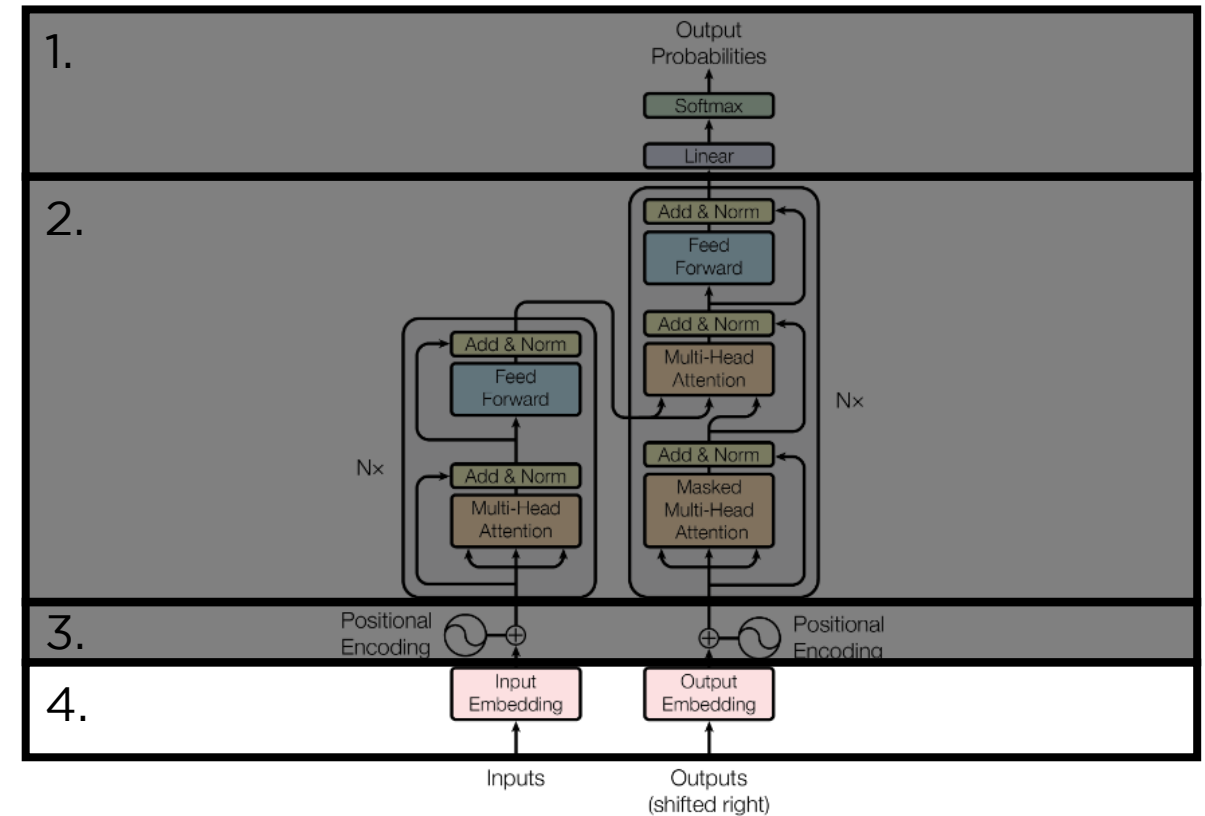


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

To constrain parameter count and extend to unseen word-forms, Transformers use sub-word tokens

Transformers: Tokenization

Transformers are designed with efficiency and parallelizability in mind, **but language modelling is inherently inefficient and non-parallelizable**

English has ~1.4M Wikitionary definitions^[1]

BERT parameters for just attention: ~86M

BERT parameters for LM: ~2,200M [theoretical under whitespace tokenization]

English is just one morphologically simple language... and we still have no way to handle unseen words

Byte-pair encodings have withstood the test of time as a strong non-parametric tokenization scheme

Transformers: Tokenization

From NLP1: words are non-atomic compounds of morphemes. So, **break words into sub-word units**. But at what granularity?

- Words inflate vocabulary size, unknown words are simply OOV
- Morphemes difficult to find
- Characters inflate sequence length

Byte-pair Encodings (BPE)^[1, 2] strikes a compromise with controlled vocabulary size

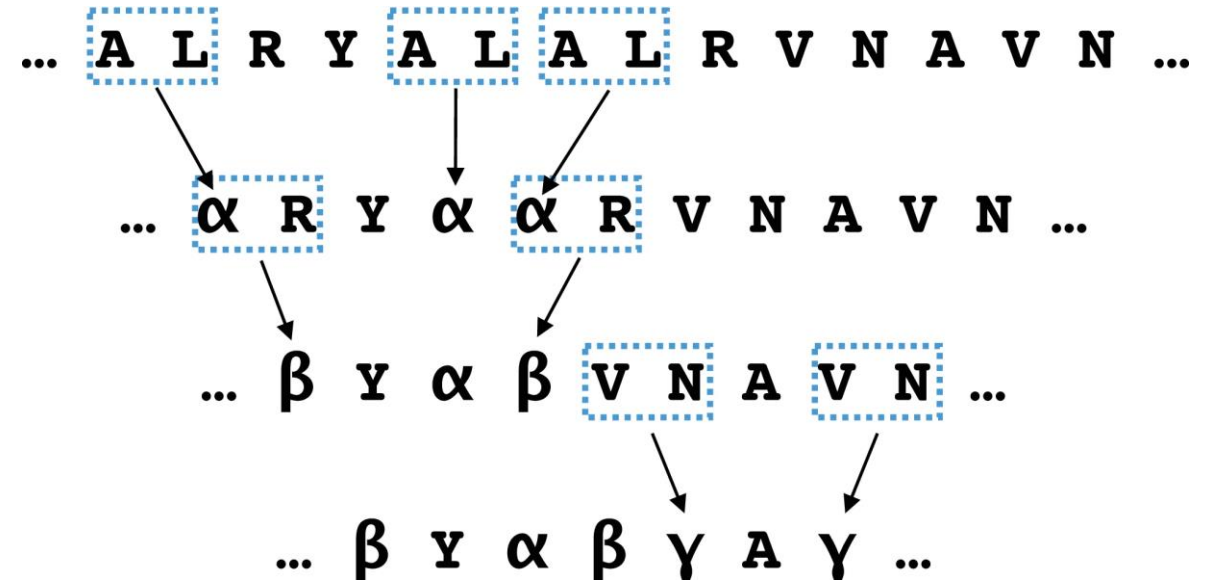


Image taken from: Kawano, K., Koide, S., & Imamura, C. (2019). Seq2seq fingerprint with byte-pair encoding for predicting changes in protein stability upon single point mutation. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(5), 1762-1772.

Byte-pair encodings handle OOV word-forms, as long as the characters have been seen during training

Transformers: Tokenization

Vocabulary can now extend to unknown word-forms as long as the characters are present in the BPE training corpus

Transformer and BERT have vocabulary of **only ~30k tokens**

BERT parameters for LM: ~24M

BERT parameters for just attention: ~86M

"Hello, y'all! How are you 😊 ?"

↓

<Hello> <,> <y'> <all> <!> <How> <are> <you>
<[UNK]> <?>

Transformer building blocks are understood, now a big picture look

Transformers: Properties

Transformers building blocks:

1. ~~Language Modelling head~~
2. ~~Multi-head scaled dot-product attention mechanisms~~
 1. ~~Residual connections (Add)~~
 2. ~~Layer normalization (Norm)~~
 3. ~~Feed forward networks~~
3. ~~Positional embeddings~~
4. ~~Word-piece embeddings~~

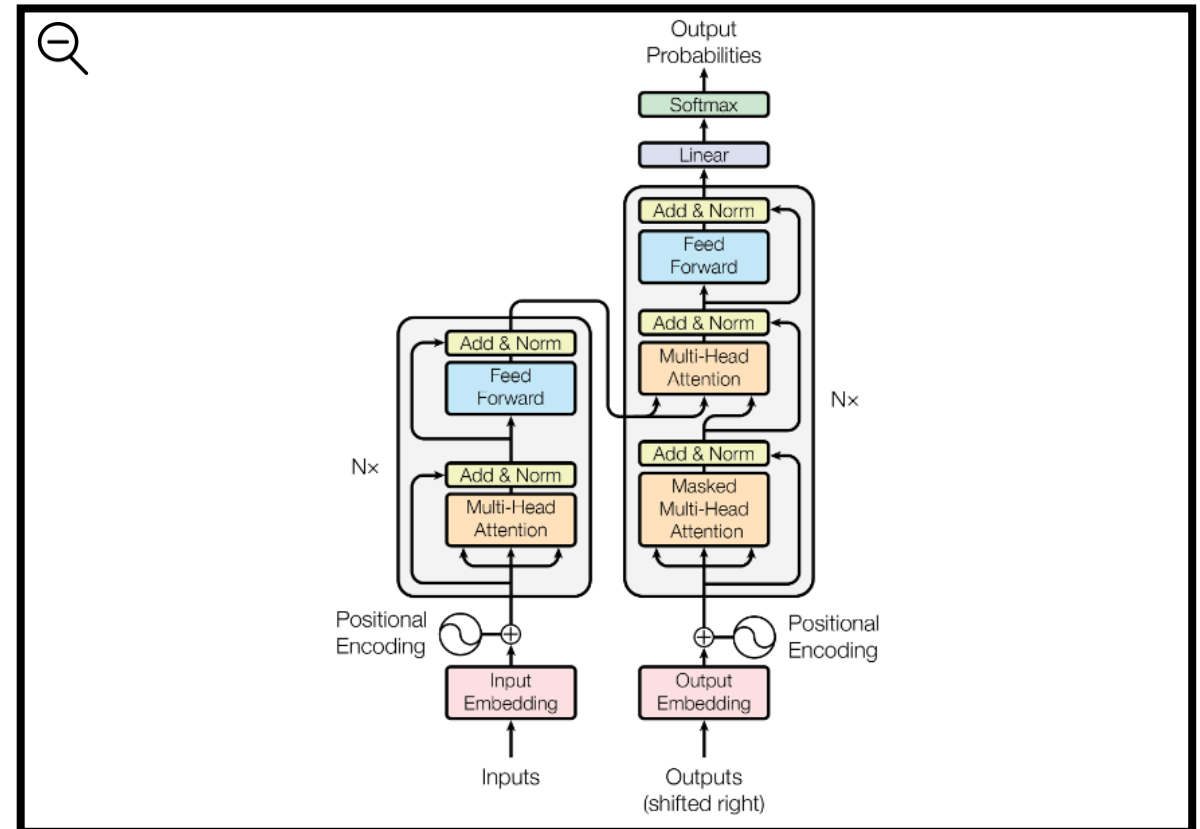


Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

RNNs naturally lend themselves to modelling language, Transformers not so much

Transformers: Properties

RNNs

1. Follow word order
2. Naturally recurrent, can be made recursive
3. Memory scales linearly with sequence length
4. Do not scale with depth
5. Expressive, handles formal languages¹

RNNs naturally lend themselves to modelling language, Transformers not so much

Transformers: Properties

RNNs

1. Follow word order
2. Naturally recurrent, can be made recursive
3. Memory scales linearly with sequence length
4. Do not scale with depth
5. Expressive, handles formal languages¹

Transformers

1. Have no real notion of word order
(permutation equivariance of attention)
2. Have no inductive bias towards recurrent or recursive structures
3. Incur quadratic memory scaling with sequence length
4. Theoretically collapse under depth
5. Fail on simple formal languages¹

RNNs naturally lend themselves to modelling language, Transformers not so much

Transformers: Properties

Transformers work.

Transformers are carefully engineered to address issues with RNN architectures

Transformers: Properties

In theory computationally heavy, but every element is designed to **avoid recurrence** and **enable parallelization**, leveraging GPUs

Plus, by connecting all tokens to each other, *should* be able to handle longer sequences

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformers are carefully engineered to address issues with RNN architectures

Transformers: Properties

In theory computationally heavy, but every element is designed to **avoid recurrence** and **enable parallelization**, leveraging GPUs

Plus, by connecting all tokens to each other, *should* be able to handle longer sequences

Achieves **better results at 300x lower training cost**, and proves (again and again) that performance scales with model and dataset size

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length	
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$	
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$	
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$	
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$	

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	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

The discussed drawbacks of Transformers can be improved or are less important

Transformers: Properties

Transformers (in theory)

- ~~1. Have no real notion of word order
(permutation equivariance of attention)~~
2. Have no inductive bias towards recurrent or recursive structures
3. Incur quadratic memory scaling with sequence length
4. Theoretically collapse under depth
5. Fail on simple formal languages¹

Transformers (in practise)

1. Position encodings with residual connections add position information throughout

The discussed drawbacks of Transformers can be improved or are less important

Transformers: Properties

Transformers (in theory)

1. ~~Have no real notion of word order
(permutation equivariance of attention)~~
2. ~~Have no inductive bias towards recurrent or
recursive structures~~
3. Incur quadratic memory scaling with
sequence length
4. Theoretically collapse under depth
5. Fail on simple formal languages¹

Transformers (in practise)

1. Position encodings with residual connections
add position information throughout
2. Can be added (see Universal Transformer¹ or
RSA²)

The discussed drawbacks of Transformers can be improved or are less important

Transformers: Properties

Transformers (in theory)

- ~~1. Have no real notion of word order
(permutation equivariance of attention)~~
- ~~2. Have no inductive bias towards recurrent or recursive structures~~
- ~~3. Incur quadratic memory scaling with sequence length~~
4. Theoretically collapse under depth
5. Fail on simple formal languages¹

Transformers (in practise)

1. Position encodings with residual connections add position information throughout
2. Can be added (see Universal Transformer¹ or RSA²)
3. Easier to train due to parallelizability
(+efficient transformers linearize cost)

The discussed drawbacks of Transformers can be improved or are less important

Transformers: Properties

Transformers (in theory)

- ~~1. Have no real notion of word order
(permutation equivariance of attention)~~
- ~~2. Have no inductive bias towards recurrent or recursive structures~~
- ~~3. Incur quadratic memory scaling with sequence length~~
- ~~4. Theoretically collapse under depth~~
5. Fail on simple formal languages¹

Transformers (in practise)

1. Position encodings with residual connections add position information throughout
2. Can be added (see Universal Transformer¹ or RSA²)
3. Easier to train due to parallelizability
(+efficient transformers linearize cost)
4. Proven to scale to very deep architectures

The discussed drawbacks of Transformers can be improved or are less important

Transformers: Properties

Transformers (in theory)

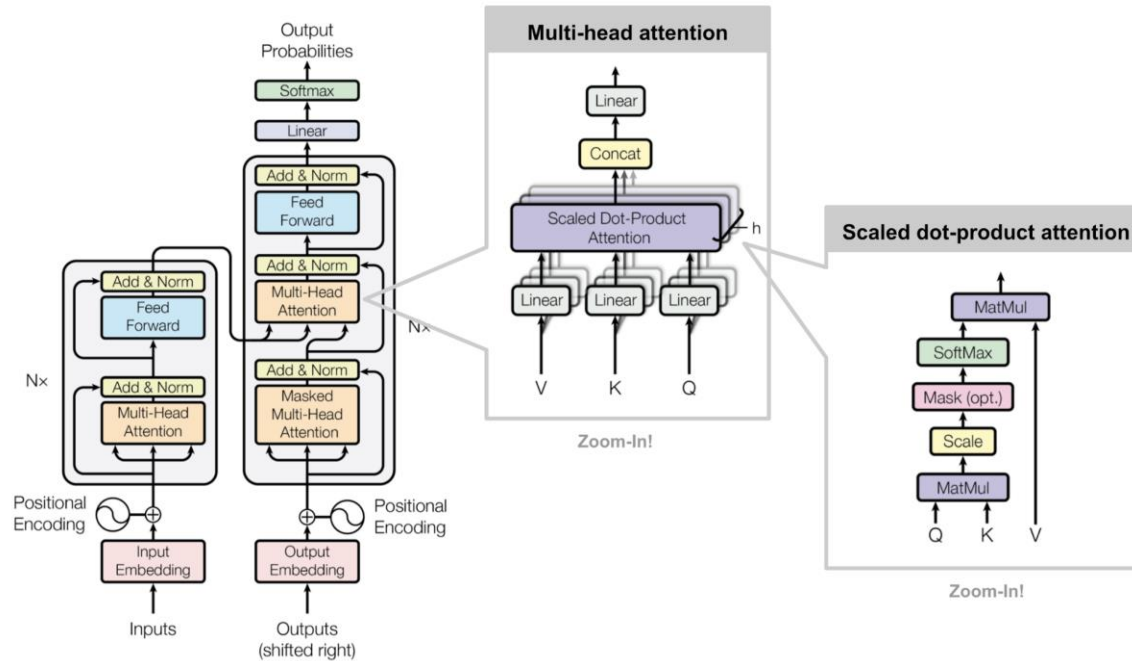
- ~~1. Have no real notion of word order
(permutation equivariance of attention)~~
- ~~2. Have no inductive bias towards recurrent or recursive structures~~
- ~~3. Incur quadratic memory scaling with sequence length~~
- ~~4. Theoretically collapse under depth~~
- ~~5. Fail on simple formal languages¹~~

Transformers (in practise)

1. Position encodings with residual connections add position information throughout
2. Can be added (see Universal Transformer¹ or RSA²)
3. Easier to train due to parallelizability (+efficient transformers linearize cost)
4. Proven to scale to very deep architectures
5. We care about *natural* language

The “Attention is All You Need” paper should hold no more surprises for you

Transformers: Closing Remarks



	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

Image taken from: <https://lilianweng.github.io/posts/2018-06-24-attention/>

Image taken from: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

The “Attention is All You Need” paper should hold no more surprises for you

Tips & Tricks: Introduction

But training out-of-the-box will probably not work.

Learning rate warmup is critical for pre-training, sometimes important for fine-tuning

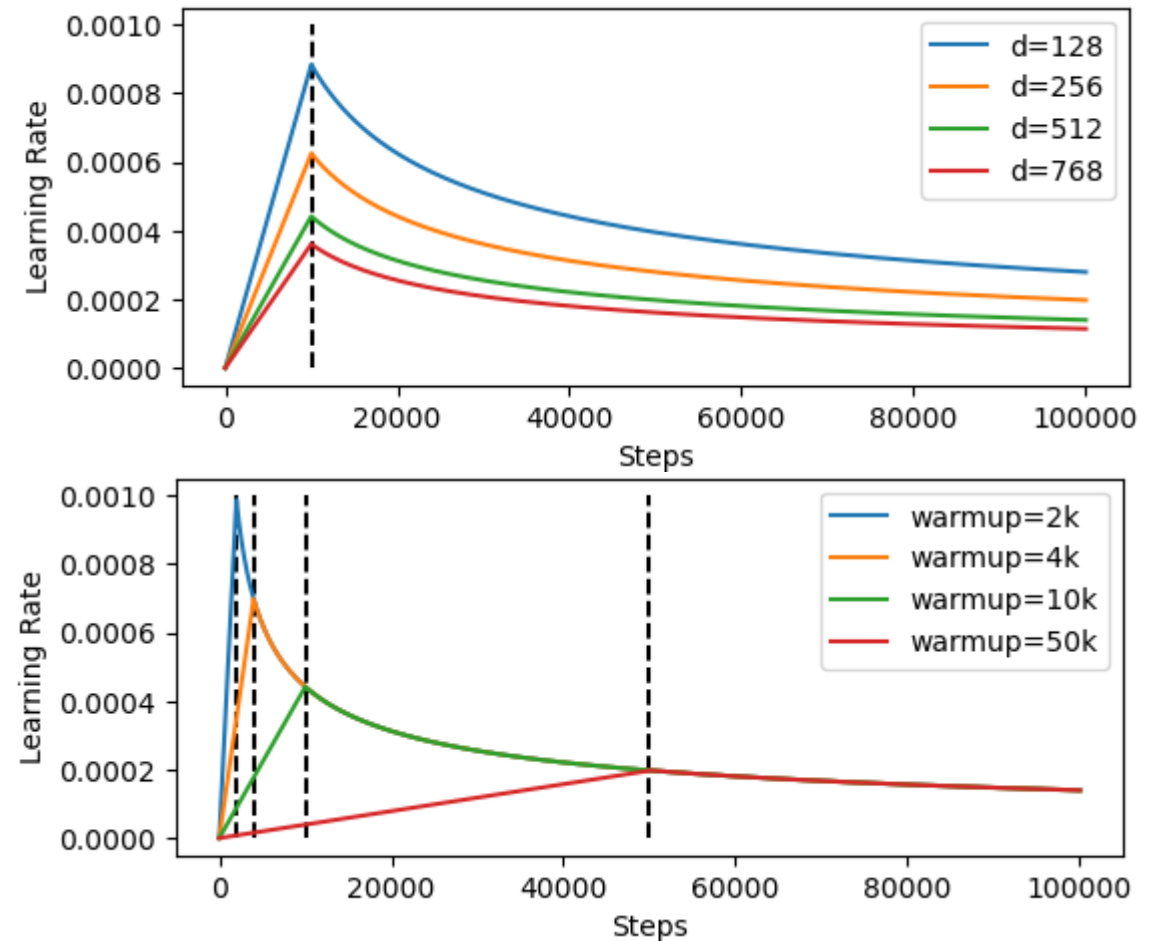
Tips & Tricks: Learning Rate Schedule

Paper uses as initial learning rate $\frac{1}{\sqrt{D_Q}}$

Scheduled using cosine annealed learning rate decay after linear warmup of 4%

Warmup mitigates effect of early (mis)steps

Sometimes necessary when finetuning



Learning rate warmup is critical for pre-training, sometimes important for fine-tuning

Tips & Tricks: Learning Rate Schedule

Transformers see large initial updates, before practically stopping: stuck in bad local optima

Use a different optimizer (e.g. RAdam^[1]) or changing order normalization and residual (pre-LN). AdamW with warmup remains default

A different strategy (also reduces training time) is sentence-length warmup (short to long)^[2]

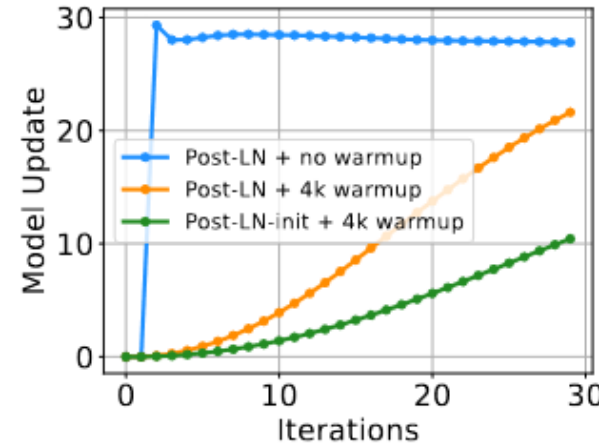


Image taken from: Wang, H., Ma, S., Dong, L., Huang, S., Zhang, D., & Wei, F. (2022). Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*.

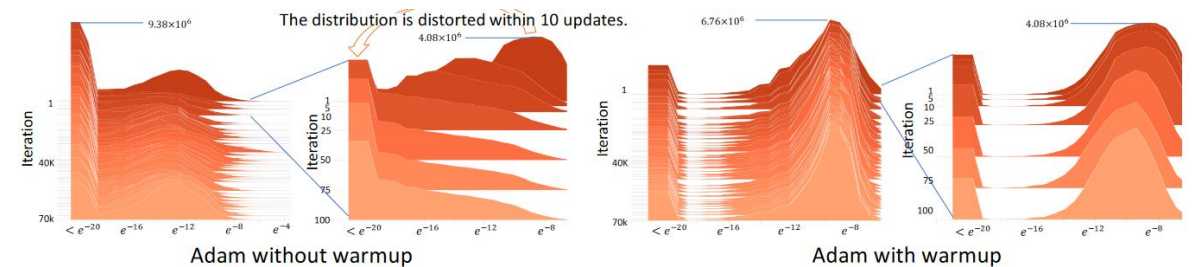


Image taken from: Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Fine-tuning with pre-trained Transformer backbone still requires hyperparameter tuning

Tips & Tricks: Learning Rate Schedule

Some findings of: Dodge et al. (2020)^[1]

High variance in random seed, for both weight initialization and data order. There is a **best** seed (they suggest 12 for CLF head)

Early performance is indicative of final performance. So **“start many, stop early and continue some”**

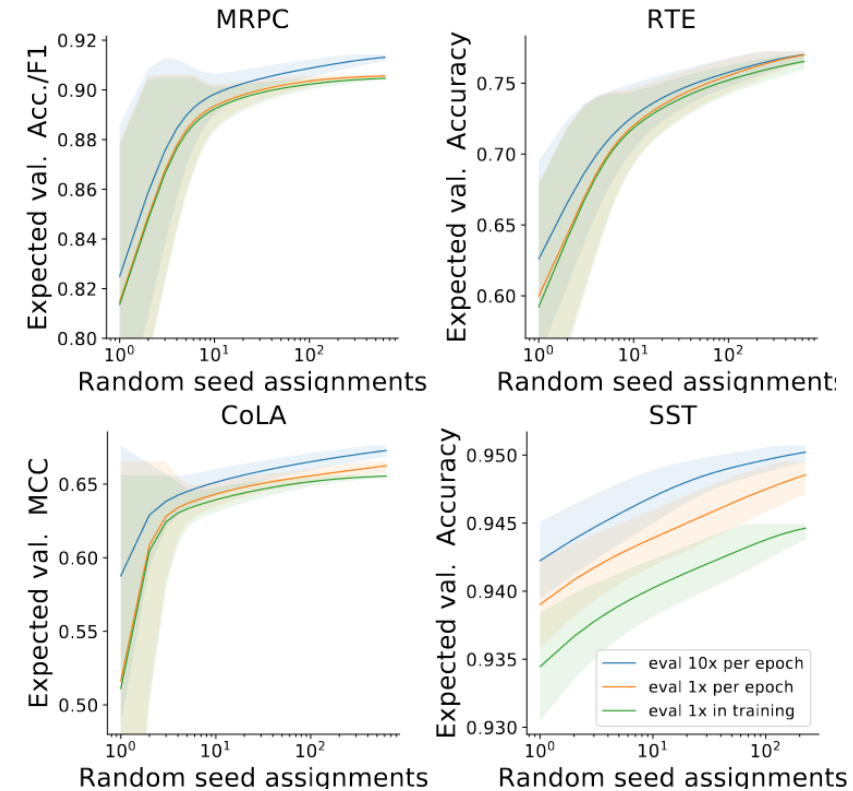


Image taken from: Dodge, Jesse, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping." *arXiv preprint arXiv:2002.06305* (2020).

Fine-tuning with pre-trained Transformer backbone still requires hyperparameter tuning

Tips & Tricks: Learning Rate Schedule

Some findings of: Zhang et al. (2021)^[1]

Int. Task: before small dataset finetuning, first move to large(ish) intermediate task (e.g. MNLI)

Re-init: re-initialise top layers and train with new classification head, just not too many

LLRD: decay learning rate with model depth

	Standard	Int. Task	LLRD	Mixout	Pre-trained WD	WD	Re-init	Longer
RTE	69.5 ± 2.5	<u>81.8 ± 1.7</u>	69.7 ± 3.2	<u>71.3 ± 1.4</u>	69.6 ± 2.1	69.5 ± 2.5	<u>72.6 ± 1.6</u>	<u>72.3 ± 1.9</u>
MRPC	90.8 ± 1.3	<u>91.8 ± 1.0</u>	91.3 ± 1.1	90.4 ± 1.4	90.8 ± 1.3	90.8 ± 1.3	<u>91.4 ± 0.8</u>	91.0 ± 1.3
STS-B	89.0 ± 0.6	89.2 ± 0.3	<u>89.2 ± 0.4</u>	<u>89.2 ± 0.4</u>	89.0 ± 0.5	89.0 ± 0.6	<u>89.4 ± 0.2</u>	<u>89.6 ± 0.3</u>
CoLA	63.0 ± 1.5	<u>63.9 ± 1.8</u>	63.0 ± 2.5	61.6 ± 1.7	63.4 ± 1.5	63.0 ± 1.5	<u>64.2 ± 1.6</u>	62.4 ± 1.7

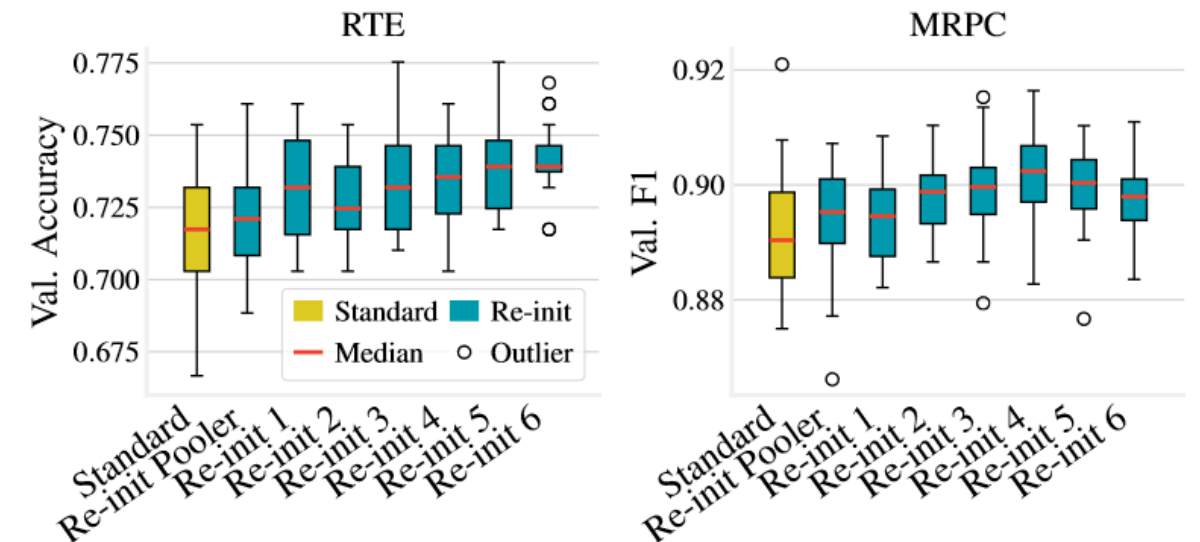


Image taken from: Zhang, T., Wu, F., Katiyar, A., Weinberger, K. Q., & Artzi, Y. (2020). Revisiting few-sample BERT fine-tuning. *arXiv preprint arXiv:2006.05987*.

Fine-tuning with pre-trained Transformer backbone still requires hyperparameter tuning

Tips & Tricks: Learning Rate Schedule

Some findings of: Zhang et al. (2021)^[1] & Mosbach et al. (2021)^[2]

Small learning rate, **with bias corrected Adam**
(BERT paper suggests without)

Train for long periods, let training loss converge
(although most gains made early)

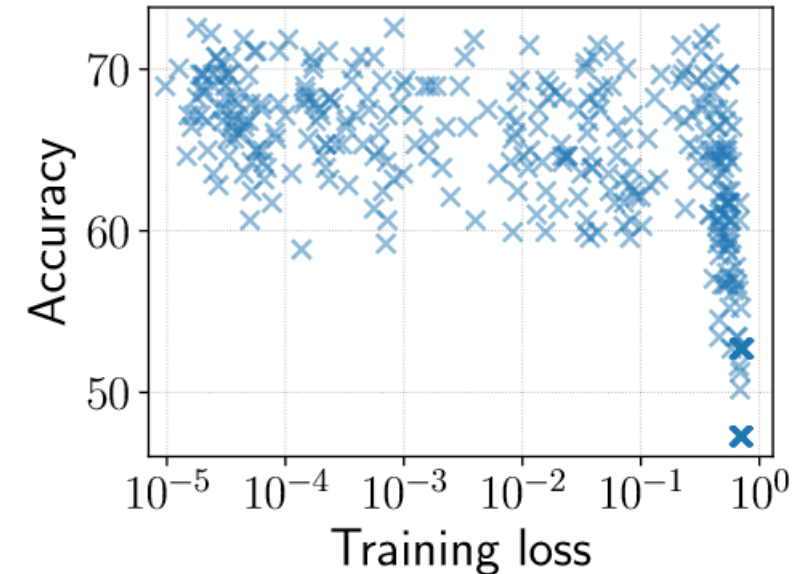


Image taken from: Mosbach, M., Andriushchenko, M., & Klakow, D. (2020). On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

Main constraint is data

Tips & Tricks: Introduction

Transformers are data hungry

Pre-training occurs over period of days on billions of tokens

Fine-tuning hyperparameters are primarily a concern in low data regime

Table of contents - Revisited

Outro

Attention	A recipe for attention mechanisms, important properties and some connections
Examples	A whirlwind tour of attention mechanisms as found throughout NLP and ML applications
Transformers	Deep dive on the Transformer architecture: embeddings, self-attention layers, encoder-decoder stacks and vocabulary generation
Tips & Tricks	Necessary tricks of the trade when training and fine-tuning transformers

The end.

Outro





UNIVERSITY
OF AMSTERDAM



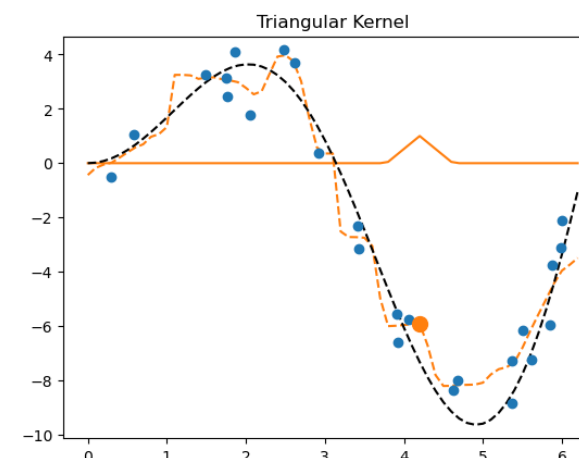
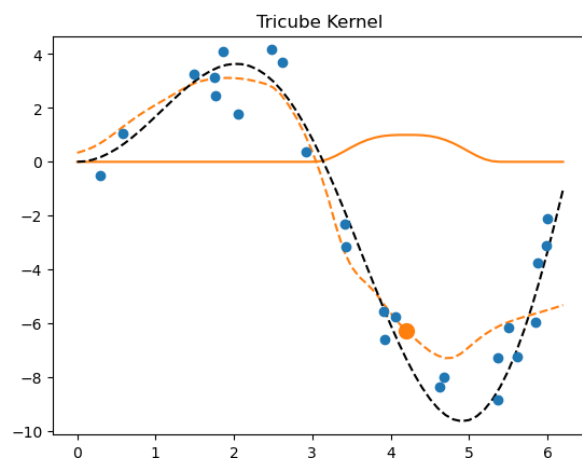
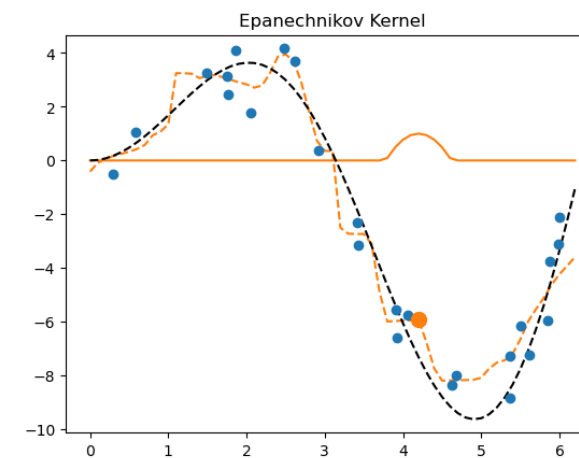
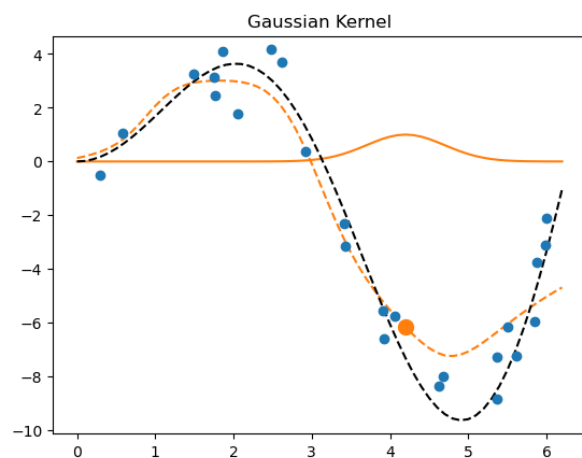
INSTITUTE FOR LOGIC,
LANGUAGE AND COMPUTATION

Attention & Transformers

APPENDIX

The form of kernel has a strong effect on the output representations

Attention: Kernel Regression



Dot-product attention is closely related to kernel regression with a log-Gaussian kernel

Attention: Kernel Regression

Recall the comparison of kernel regression?

Kernel regression with a Gaussian kernel is a pretty close analogy to scaled dot-product attention

$$\kappa(\mathbf{q}, \mathbf{k}) = -\frac{1}{2} \|\mathbf{q} - \mathbf{k}\|_2^2$$

Dot-product attention is closely related to kernel regression with a log-Gaussian kernel

Attention: Kernel Regression

Recall the comparison of kernel regression?

Kernel regression with a Gaussian kernel is a pretty close analogy to scaled dot-product attention

Expand the quadratic form

$$\begin{aligned}\kappa(\mathbf{q}, \mathbf{k}) &= -\frac{1}{2} \|\mathbf{q} - \mathbf{k}\|_2^2 \\ &= -\frac{1}{2} (\|\mathbf{q}\|_2^2 - 2\mathbf{q}^\top \mathbf{k} + \|\mathbf{k}\|_2^2) \\ &= \mathbf{q}^\top \mathbf{k} - \frac{1}{2} \|\mathbf{q}\|_2^2 - \frac{1}{2} \|\mathbf{k}\|_2^2\end{aligned}$$

Dot-product attention is closely related to kernel regression with a log-Gaussian kernel

Attention: Kernel Regression

Recall the comparison of kernel regression?

Kernel regression with a Gaussian kernel is a pretty close analogy to scaled dot-product attention

Expand the quadratic form

We softmax over \mathbf{q} , and softmax is invariant to translation

$$\begin{aligned}\kappa(\mathbf{q}, \mathbf{k}) &= -\frac{1}{2} \|\mathbf{q} - \mathbf{k}\|_2^2 \\ &= -\frac{1}{2} (\|\mathbf{q}\|_2^2 - 2\mathbf{q}^\top \mathbf{k} + \|\mathbf{k}\|_2^2) \\ &= \mathbf{q}^\top \mathbf{k} - \cancel{\frac{1}{2} \|\mathbf{q}\|_2^2} - \frac{1}{2} \|\mathbf{k}\|_2^2\end{aligned}$$

Dot-product attention is closely related to kernel regression with a log-Gaussian kernel

Attention: Kernel Regression

Recall the comparison of kernel regression?

Kernel regression with a Gaussian kernel is a pretty close analogy to scaled dot-product attention

Expand the quadratic form

We softmax over \mathbf{q} , and softmax is invariant to translation

Assume $\mathbf{k} \sim \mathcal{N}(0, 1)$, e.g. through layer or batch norm, such that $\|\mathbf{k}\|_2^2 \approx \text{constant}$, see above point

$$\begin{aligned}
 \kappa(\mathbf{q}, \mathbf{k}) &= -\frac{1}{2} \|\mathbf{q} - \mathbf{k}\|_2^2 \\
 &= -\frac{1}{2} (\|\mathbf{q}\|_2^2 - 2\mathbf{q}^\top \mathbf{k} + \|\mathbf{k}\|_2^2) \\
 &= \mathbf{q}^\top \mathbf{k} - \cancel{\frac{1}{2} \|\mathbf{q}\|_2^2} - \cancel{\frac{1}{2} \|\mathbf{k}\|_2^2}
 \end{aligned}$$

Dot-product attention is closely related to kernel regression with a log-Gaussian kernel

Attention: Kernel Regression

Recall the comparison of kernel regression?

Kernel regression with a Gaussian kernel is a pretty close analogy to scaled dot-product attention

Kernel currently assumes $\sigma = 1$, if we allowed it to vary, we would recover scaled dot-product attention

$$\begin{aligned}
 \kappa(\mathbf{q}, \mathbf{k}) &= -\frac{1}{2} \|\mathbf{q} - \mathbf{k}\|_2^2 \\
 &= -\frac{1}{2} (\|\mathbf{q}\|_2^2 - 2\mathbf{q}^\top \mathbf{k} + \|\mathbf{k}\|_2^2) \\
 &= \mathbf{q}^\top \mathbf{k} - \cancel{\frac{1}{2} \|\mathbf{q}\|_2^2} - \cancel{\frac{1}{2} \|\mathbf{k}\|_2^2}
 \end{aligned}$$

Position embeddings are still absolute, relative embeddings and attention has been shown to help

Transformers: Position Encoding

Learning separate embeddings for a (limited) set of positions shows substantial improvement in results

Essentially treats attention as relational edge weights: graph is no longer fully connected, or homogenous

DeBERTa takes this concept further and shows substantial gains over other encoder only models^[1]

Model	Position Information	EN-DE BLEU	EN-FR BLEU
Transformer (base)	Absolute Position Representations	26.5	38.2
Transformer (base)	Relative Position Representations	26.8	38.7
Transformer (big)	Absolute Position Representations	27.9	41.2
Transformer (big)	Relative Position Representations	29.2	41.5

$$e_{ij} = \frac{x_i W^Q (x_j W^K)^T + x_i W^Q (a_{ij}^K)^T}{\sqrt{d_z}}$$

$$z_i = \sum_{j=1}^n \alpha_{ij} (x_j W^V + a_{ij}^V)$$

Image taken from: Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*.

Byte-pair encodings operate on a simple principle of iteratively merging frequent character pairs

Transformers: Tokenization

1. Count up the tokens in the corpus
2. Find the most frequent pair of characters/symbols
3. Update the vocabulary
4. Repeat 2 until vocabulary size is as desired

Strings	Count
<u>pu</u> n	12
h u g	10
<u>pu</u> g	5
b u n s	5

Most frequent character pair
(`'p'`, `'u'`) 17

Byte-pair encodings operate on a simple principle of iteratively merging frequent character pairs

Transformers: Tokenization

1. Count up the tokens in the corpus
2. Find the most frequent pair of characters/symbols
3. Update the vocabulary
4. Repeat 2 until vocabulary size is as desired

Strings	Count
<u>pun</u>	12
h u g	10
pu g	5
b u n s	5

Most frequent character pair

('p', 'u') 17

('pu', 'n') 12

Byte-pair encodings operate on a simple principle of iteratively merging frequent character pairs

Transformers: Tokenization

1. Count up the tokens in the corpus
2. Find the most frequent pair of characters/symbols
3. Update the vocabulary
4. Repeat 2 until vocabulary size is as desired

Strings	Count
pun	12
<u>h</u> u g	10
pu g	5
b u n s	5

Most frequent character pair

('p', 'u') 17

('pu', 'n') 12

('h', 'u') 10

Byte-pair encodings operate on a simple principle of iteratively merging frequent character pairs

Transformers: Tokenization

1. Count up the tokens in the corpus
2. Find the most frequent pair of characters/symbols
3. Update the vocabulary
4. Repeat 2 until vocabulary size is as desired

Strings	Count
pun	12
hug	10
pug	5
<u>bu</u> n s	5

Most frequent character pair

```
('p', 'u') 17
('pu', 'n') 12
('h', 'u') 10
('hu', 'g') 10
('pu', 'g') 5
('b', 'u') 5
```

Learning rate warmup is critical for pre-training, sometimes important for fine-tuning

Tips & Tricks: Learning Rate Schedule

During pre-training, very long (~50%) warmup periods followed by aggressive learning rate annealing shows promise

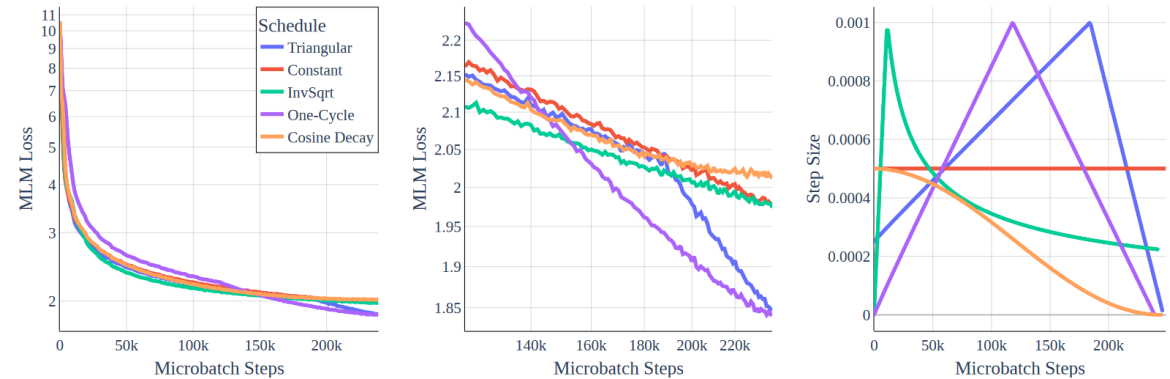


Image taken from: Geiping, J., & Goldstein, T. (2022). Cramming: Training a Language Model on a Single GPU in One Day. *arXiv preprint arXiv:2212.14034*.

Transformers are designed with GPUs in mind, utilize them as much as possible

Tips & Tricks: Learning Rate Schedule

As long as learning rate is adjusted, batch size (at equal budget) correlates with performance

If you can, use mixed precision training. Large speedup with no performance cost

One weird Twitter trick, pad the vocabulary dimension to the nearest multiple of 8 for 25% speedup for free

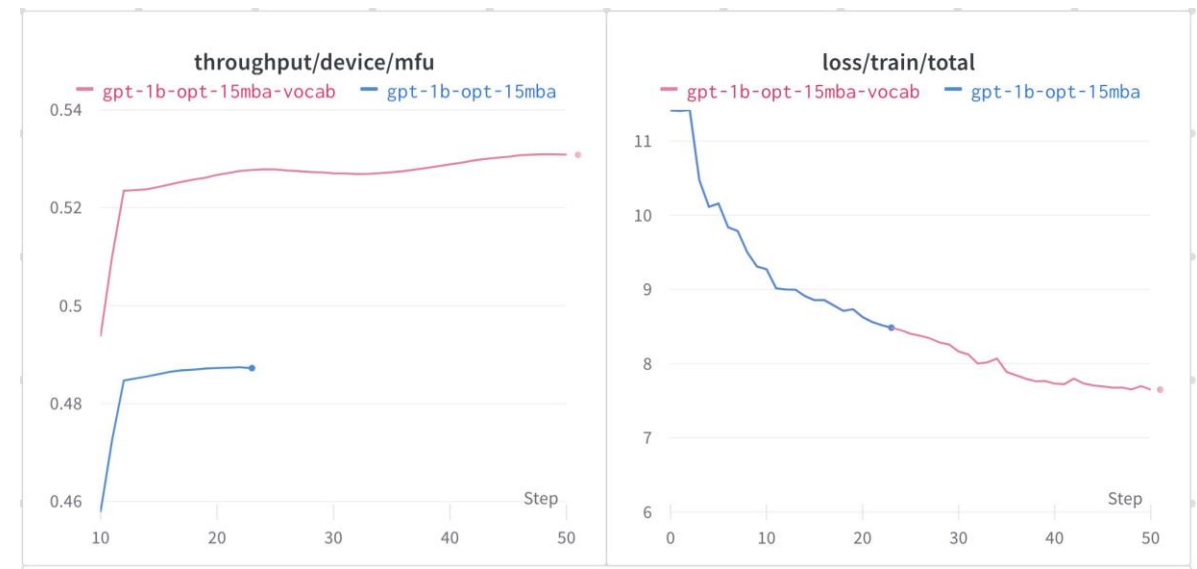


Image taken from: https://twitter.com/abhi_venigalla/status/1621710130051190784/photo/1