Transformers

Vijay Anand Raghavan

Agenda

1. Attention
2. Transformer Architecture
3. Causal Mediation Analysis
4. Transformers and Memory
5. Memory Editing in Transformers

The technical problem

- Gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization).
- Even if we assume that the parameters are such that the recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians)

Challenges with RNN Hidden States

- Encoding entire past sequence is inefficient
- Most context not relevant for next step prediction
- Analogous to keeping every previous physical state in working memory

Attention

Human Attention

- Humans focus on a few words to predict the next word
- Words focused on changes dynamically based on context
- For example: "huge green elephant" ---> focus on "huge" & "elephant"
- Want models to mimic this flexible attention

What is an Attention Mechanism?

- Attention layers establish dynamic connections between input and output
- This filtering determines which specific parts of the input to focus attention on based relevance to the current output

Attention for NLP Models

- Input: sequence of input token representations (e.g. word embeddings)
- Output: target word representation that model makes prediction for
- Attention used to retrieve relevant context, filter out irrelevant

Example

```
The pink elephant tried to get into the car but it was too
```

- The word that follows too is ‘big’ but how do we know that
- Paying attention to certain words that are important and this is what an attention head in a transformer is designed to do.

Implementing Attention

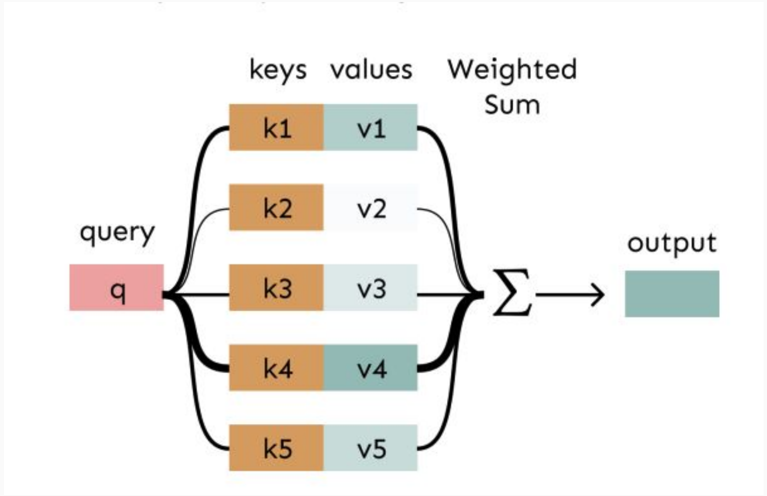
- Attention layers connect input representations to output
- Input filtered based on relevance to current output
- Attention weights represent relevance filtering
- High weights: very relevant words for this output
- Low weights: irrelevant words effectively ignored

Query, Key, Value

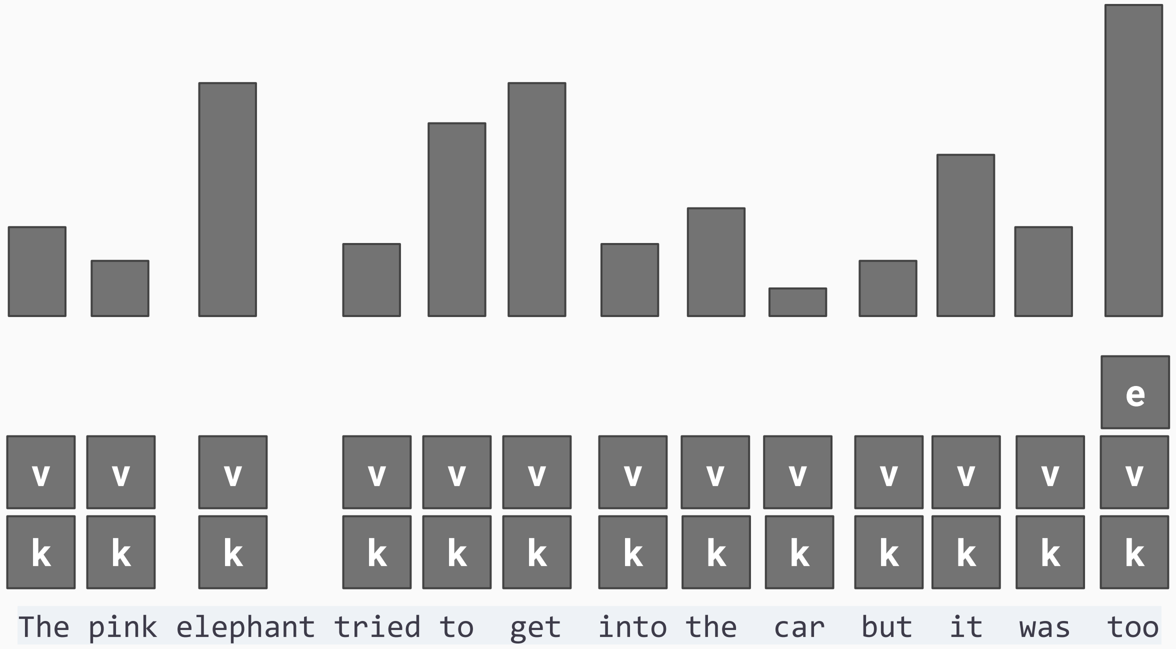
- The *Query (q)* can be thought of as a representation of the current task at hand (e.g., “What word follows *too*?”).
- The *key* vectors *(K)* are representations of each word in the sentence—you can think of these as descriptions of the kinds of prediction tasks that each word can help with.
- The *value* vectors *(V)* are also representations of the words in the sentence—you can think of these as the unweighted contributions of each word.

Attention as a soft, averaging lookup table

- *“We can think of attention as performing fuzzy lookup in a key-value store.”*
- *“In attention, the query matches all keys softly, to a weight between 0 and 1.”*
- *“The keys’ values are multiplied by the weights and summed.”*



Example - Attention weights for *'too'*



1. Words to embeddings

Let $w_{1:n}$ be a sequence of words in Vocabulary V.

For each w_i let b_i be the embedding of the word vector.

$$\begin{tikzcd}
w_1 \arrow[d] & w_2 \arrow[d] & w_3 \arrow[d] & w_4 \arrow[d] & w_5 \arrow[d] & \cdots\cdots\cdots & w_n \arrow[d] \\
b_1 & b_2 & b_3 & b_4 & b_5 & & b_n
\end{tikzcd}$$

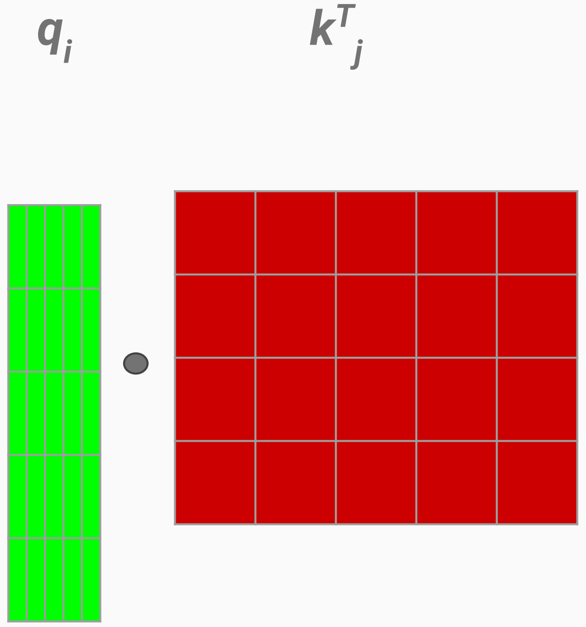
2. Transform each word embedding with weight matrix resulting in Q, K, V

- $q_i = W_i^Q b_i$
- $k_i = W_i^k b_i$
- $v_i = W_i^v b_i$

3. Compute Pairwise similarities between queries and keys

$p_{ij} = q_i k_j^T$

●	k_1	k_2	k_3	k_n
q_1	p_{11}	p_{12}	p_{13}	p_{1n}
q_2	p_{21}	p_{22}	p_{23}	p_{2n}
q_3	p_{31}	p_{32}	p_{33}	p_{3n}
q_n	p_{n1}	p_{n2}	p_{n3}	p_{nn}

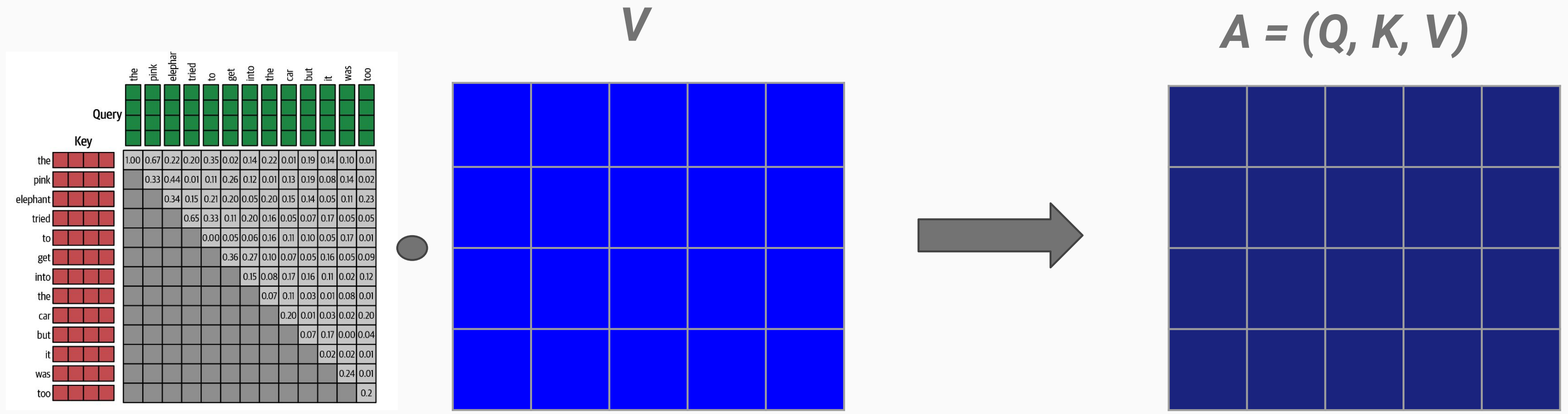


4. Normalize with softmax

$n_{ij} = exp(p_{ij}) / \sum_j exp(p_{ij}) \quad \Rightarrow \quad softmax(QK^T)$

5. Compute output for each word as weighted sum of values

$$o_i = \sum_j n_{ij} v_i$$



Key \ Query	the	pink	elephar	tried	to	get	into	the	car	but	it	was	too
the	1.00	0.67	0.22	0.20	0.35	0.02	0.14	0.22	0.01	0.19	0.14	0.10	0.01
pink		0.33	0.44	0.01	0.11	0.26	0.12	0.01	0.13	0.19	0.08	0.14	0.02
elephant			0.34	0.15	0.21	0.20	0.05	0.20	0.15	0.14	0.05	0.11	0.23
tried				0.65	0.33	0.11	0.20	0.16	0.05	0.07	0.17	0.05	0.05
to					0.00	0.05	0.06	0.16	0.11	0.10	0.05	0.17	0.01
get						0.36	0.27	0.10	0.07	0.05	0.16	0.05	0.09
into							0.15	0.08	0.17	0.16	0.11	0.02	0.12
the								0.07	0.11	0.03	0.01	0.08	0.01
car									0.20	0.01	0.03	0.02	0.20
but										0.07	0.17	0.00	0.04
it											0.02	0.02	0.01
was												0.24	0.01
too													0.2

Issue

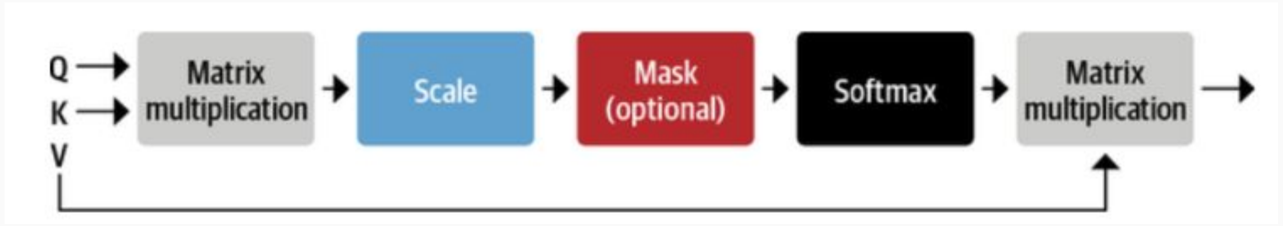
- The two most commonly used attention functions are additive attention, and dot-product (multiplicative) attention.
- Additive attention computes the compatibility function using a feed-forward network with a single hidden layer.
- While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.
- While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k.
- For large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients .

Scaling

To counteract this effect, we scale the dot products by √ 1/ dk

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

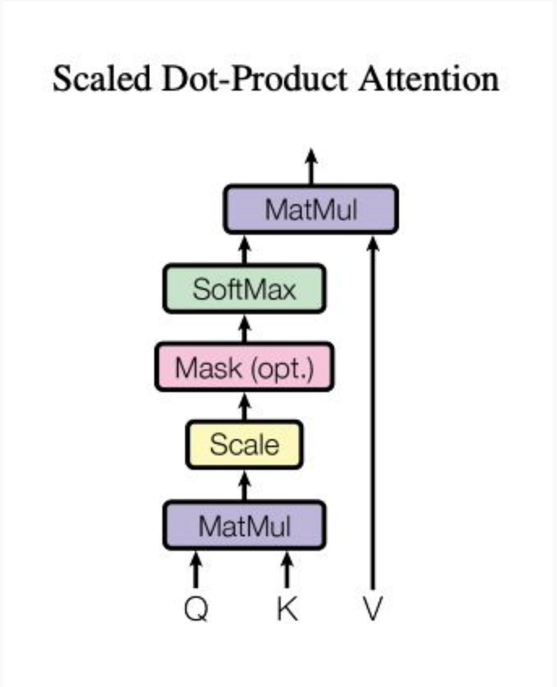
Recap - Operations in scaled dot-product attention



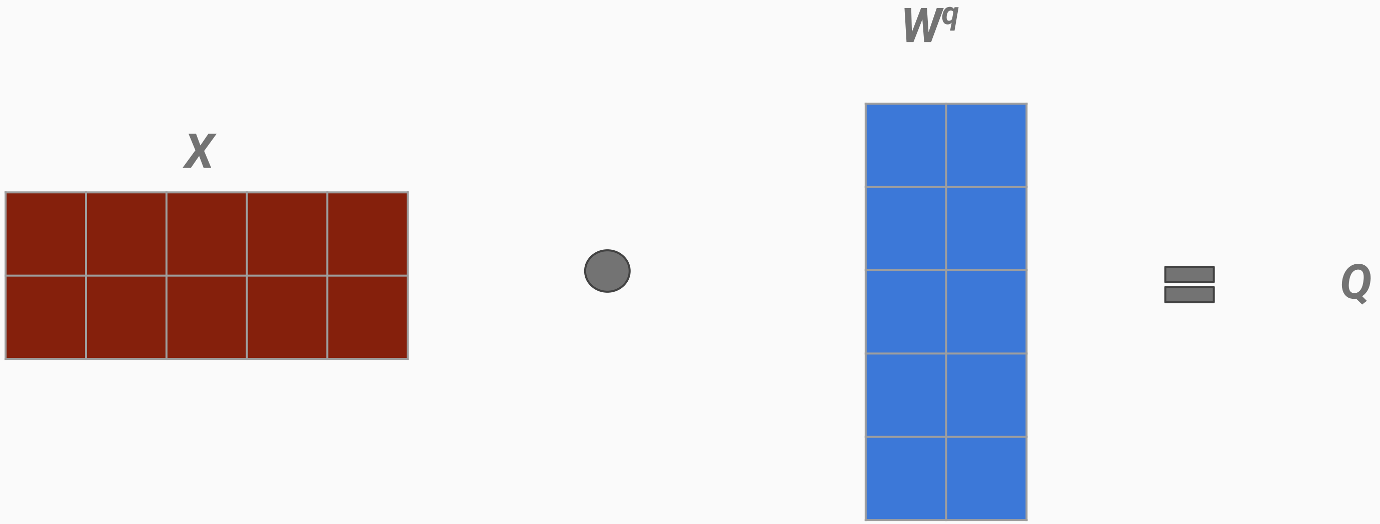
What are we learning?

So far we have just done few arithmetic operations.

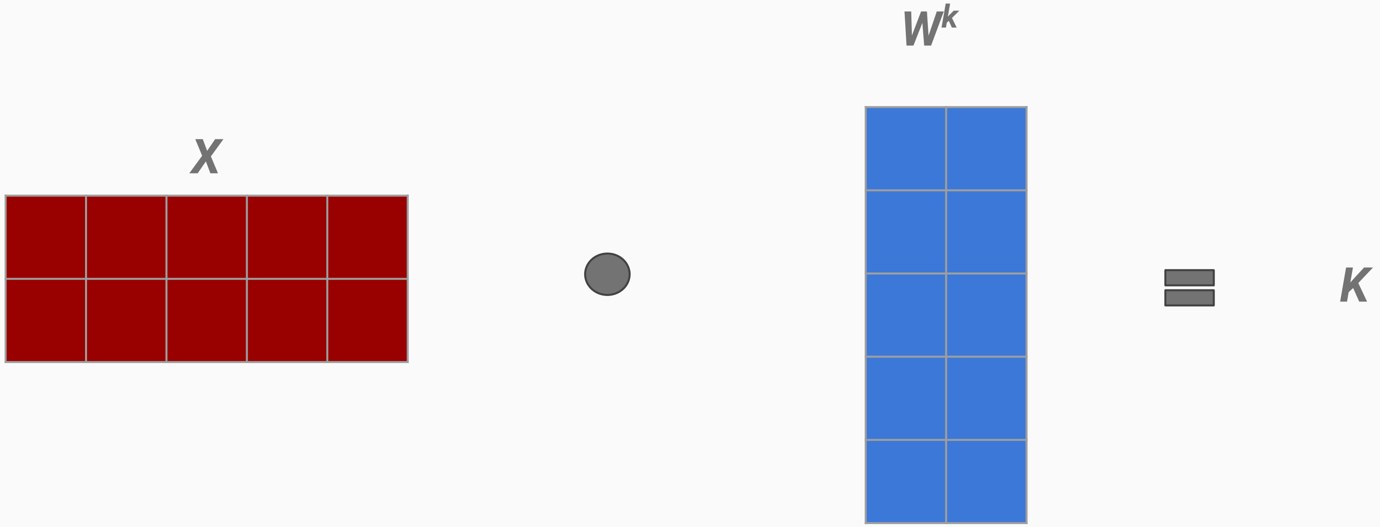
We have not learnt anything…..



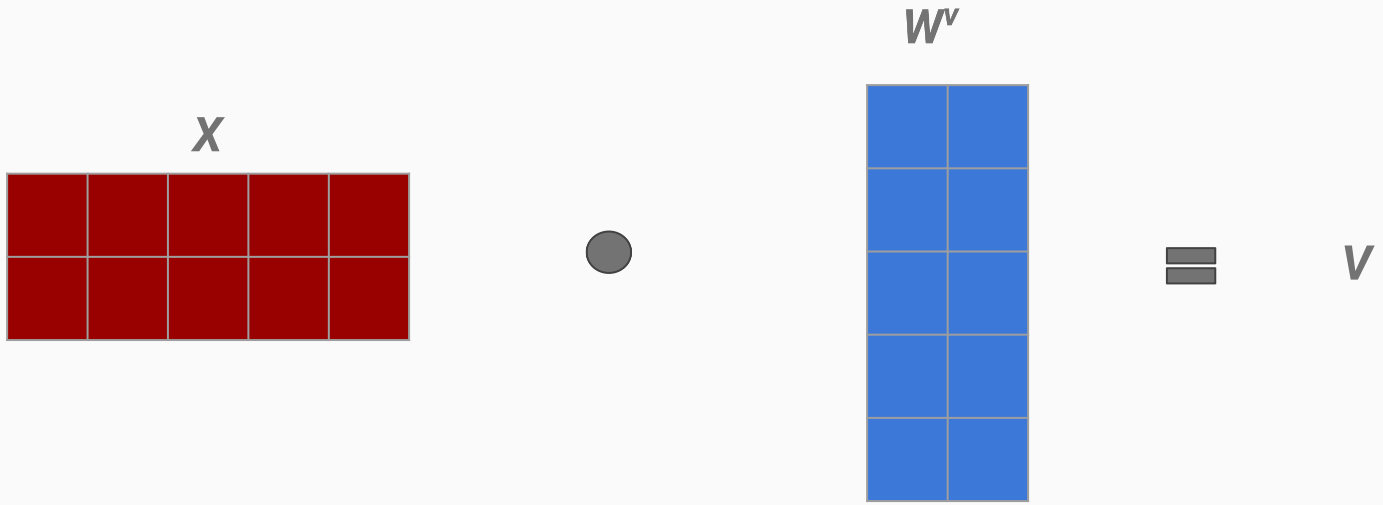
Learnable Weights W^q



Learnable Weights W^k

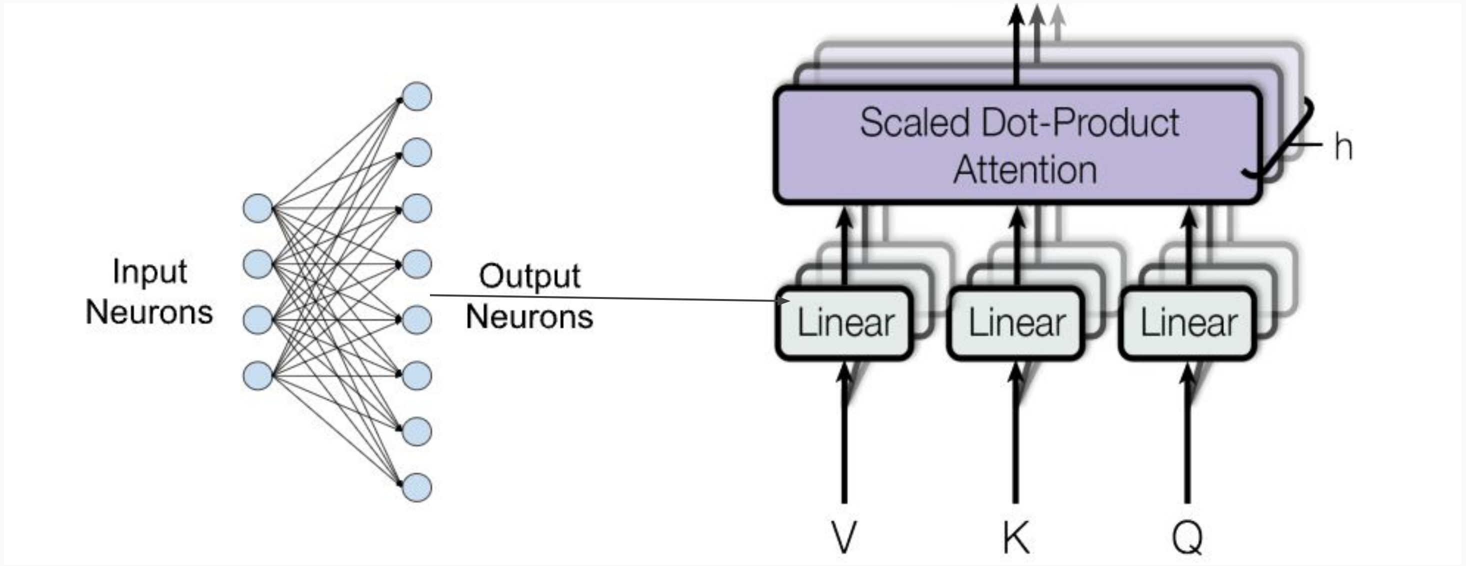


Learnable Weights W^V



Linear layer - *Example of a small fully-connected layer with four input and eight output neurons.*

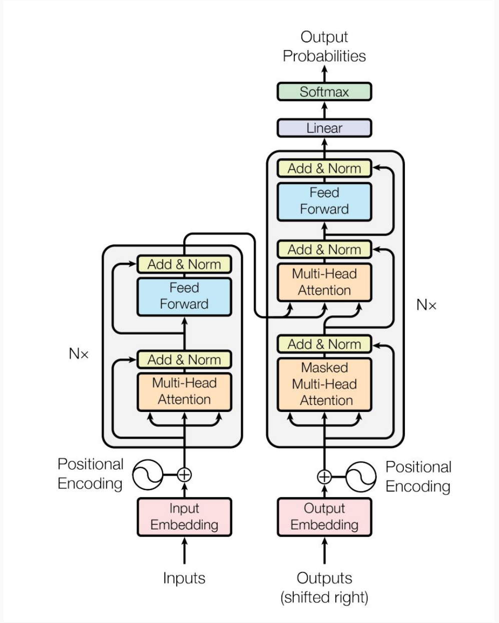
$$Attention(Q,K,V) = Attention\ (XW^Q, XW^K, XW^V)$$



https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html
Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. Advances in neural information processing systems. 2017;30.

You shall know a word by the company it keeps! J.R. Firth

- Transformers have no recurrence or convolution to exploit sequence order information.
- To make use of order, position information needs to be injected into the inputs.
- This is done by adding position encodings to the input token embeddings.
- Position encodings give each token a sense of its absolute position in the sequence.
- This allows the self-attention layers to incorporate order and distance relationships between sequence positions.



Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. Advances in neural information processing systems. 2017;30.

Positional Encoding

- For positional encoding sine and cosine functions of different frequencies are used:
  - $PE_{(pos,2i)} = sin(pos/10000^{2i/dmodel})$
  - $PE_{(pos,2i+1)} = cos(pos/10000^{2i/dmodel)})$ where *pos* is the position and *i* is the dimension.
- That is, each dimension of the positional encoding corresponds to a sinusoid.
- The wavelengths form a geometric progression from 2π to 10000 · 2π.
- This function was chosen because it would allow the model to easily learn to attend by relative positions, since for any fixed offset *k*, PE_{pos+k} can be represented as a linear function of PE_{pos}.

https://github.com/keras-team/keras-nlp/blob/v0.7.0/keras_nlp/layers/modeling/sine_position_encoding.py#L20

Sine + Cosine

$$PE_{(k,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

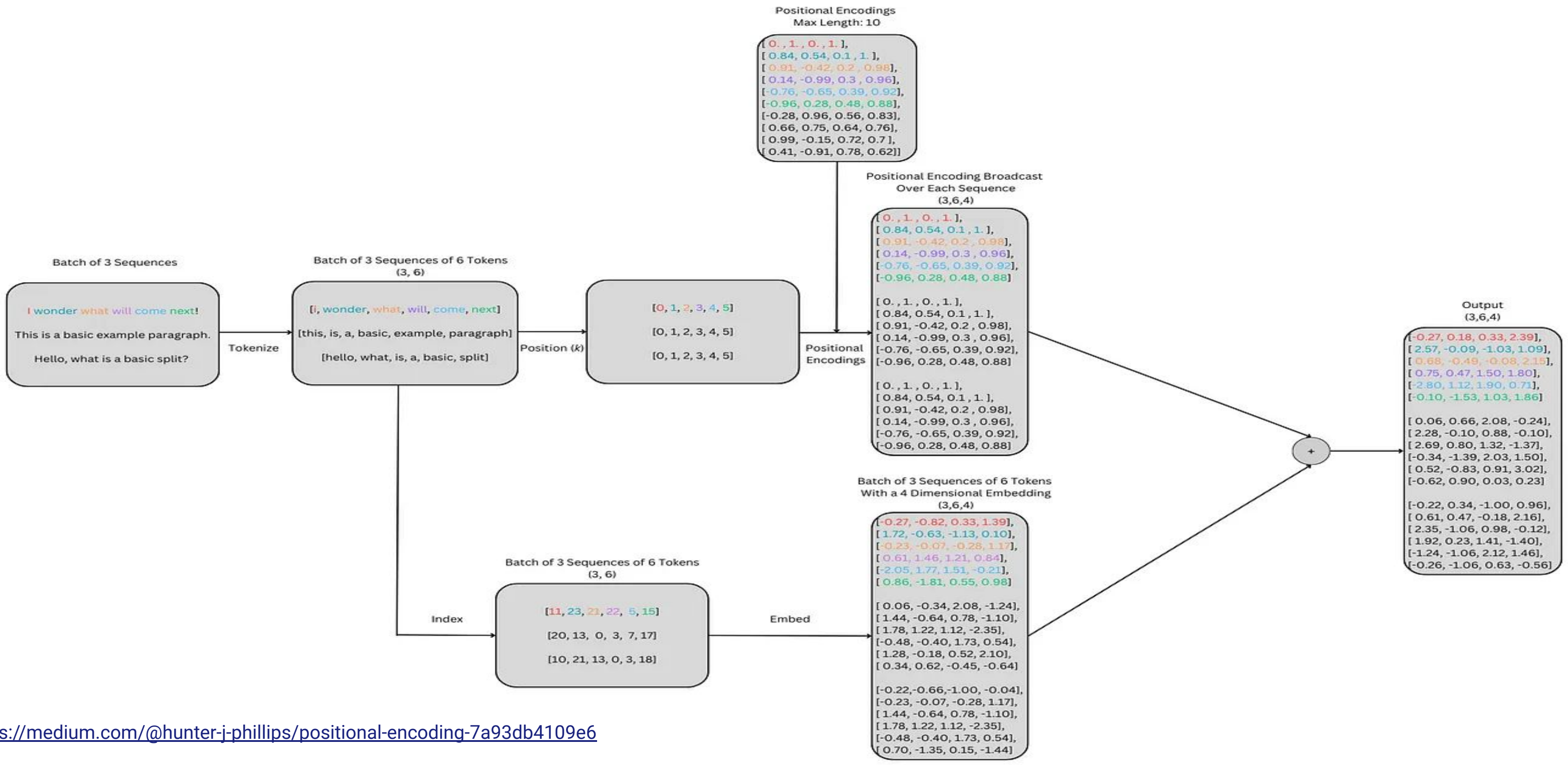
$$PE_{(k,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$k \left\{ \begin{array}{l}
P(0) = \left[\overbrace{sin(\frac{0}{10000^{\frac{0}{4}}}), cos(\frac{0}{10000^{\frac{0}{4}}}), sin(\frac{0}{10000^{\frac{1}{4}}}), cos(\frac{0}{10000^{\frac{1}{4}}})}^{d\_model = 4}\right] \\
P(1) = \left[sin(\frac{1}{10000^{\frac{0}{4}}}), cos(\frac{1}{10000^{\frac{0}{4}}}), sin(\frac{1}{10000^{\frac{1}{4}}}), cos(\frac{1}{10000^{\frac{1}{4}}})\right] \\
P(2) = \left[sin(\frac{2}{10000^{\frac{0}{4}}}), cos(\frac{2}{10000^{\frac{0}{4}}}), sin(\frac{2}{10000^{\frac{1}{4}}}), cos(\frac{2}{10000^{\frac{1}{4}}})\right] \\
P(3) = \left[sin(\frac{3}{10000^{\frac{0}{4}}}), cos(\frac{3}{10000^{\frac{0}{4}}}), sin(\frac{3}{10000^{\frac{1}{4}}}), cos(\frac{3}{10000^{\frac{1}{4}}})\right] \\
P(4) = \left[sin(\frac{4}{10000^{\frac{0}{4}}}), cos(\frac{4}{10000^{\frac{0}{4}}}), sin(\frac{4}{10000^{\frac{1}{4}}}), cos(\frac{4}{10000^{\frac{1}{4}}})\right]
\end{array}\right.$$

$$P(5) = \left[\underbrace{sin(\frac{5}{10000^{\frac{0}{4}}}), cos(\frac{5}{10000^{\frac{0}{4}}}), sin(\frac{5}{10000^{\frac{1}{4}}}), cos(\frac{5}{10000^{\frac{1}{4}}})}_{i=0 \qquad i=0 \qquad i=1 \qquad i=1}\right]$$

https://medium.com/@hunter-j-phillips/positional-encoding-7a93db4109e6

Positional Encoding with Tokens


Batch of 3 Sequences
I wonder what will come next!
This is a basic example paragraph.
Hello, what is a basic split?
Tokenize
Batch of 3 Sequences of 6 Tokens (3, 6)
[i, wonder, what, will, come, next]
[this, is, a, basic, example, paragraph]
[hello, what, is, a, basic, split]
Position (k)
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5]
Positional Encodings Max Length: 10
Positional Encodings
Positional Encoding Broadcast Over Each Sequence (3,6,4)
Index
Batch of 3 Sequences of 6 Tokens (3, 6)
[11, 23, 21, 22, 5, 15]
[20, 13, 0, 3, 7, 17]
[10, 21, 13, 0, 3, 18]
Embed
Batch of 3 Sequences of 6 Tokens With a 4 Dimensional Embedding (3,6,4)
+
Output (3,6,4)

Positional Encodings, Max Length: 10

```
[[ 0. , 1. , 0. , 1. ],
 [ 0.84, 0.54, 0.1 , 1. ],
 [ 0.91, -0.42, 0.2 , 0.98],
 [ 0.14, -0.99, 0.3 , 0.96],
 [-0.76, -0.65, 0.39, 0.92],
 [-0.96, 0.28, 0.48, 0.88],
 [-0.28, 0.96, 0.56, 0.83],
 [ 0.66, 0.75, 0.64, 0.76],
 [ 0.99, -0.15, 0.72, 0.7 ],
 [ 0.41, -0.91, 0.78, 0.62]]
```

Positional Encoding Broadcast Over Each Sequence (3,6,4)

```
[ 0. , 1. , 0. , 1. ],
[ 0.84, 0.54, 0.1 , 1. ],
[ 0.91, -0.42, 0.2 , 0.98],
[ 0.14, -0.99, 0.3 , 0.96],
[-0.76, -0.65, 0.39, 0.92],
[-0.96, 0.28, 0.48, 0.88]

[ 0. , 1. , 0. , 1. ],
[ 0.84, 0.54, 0.1 , 1. ],
[ 0.91, -0.42, 0.2 , 0.98],
[ 0.14, -0.99, 0.3 , 0.96],
[-0.76, -0.65, 0.39, 0.92],
[-0.96, 0.28, 0.48, 0.88]

[ 0. , 1. , 0. , 1. ],
[ 0.84, 0.54, 0.1 , 1. ],
[ 0.91, -0.42, 0.2 , 0.98],
[ 0.14, -0.99, 0.3 , 0.96],
[-0.76, -0.65, 0.39, 0.92],
[-0.96, 0.28, 0.48, 0.88]
```

Batch of 3 Sequences of 6 Tokens With a 4 Dimensional Embedding (3,6,4)

```
[-0.27, -0.82, 0.33, 1.39],
[ 1.72, -0.63, -1.13, 0.10],
[-0.23, -0.07, -0.28, 1.17],
[ 0.61, 1.46, 1.21, 0.84],
[-2.05, 1.77, 1.51, -0.21],
[ 0.86, -1.81, 0.55, 0.98]

[ 0.06, -0.34, 2.08, -1.24],
[ 1.44, -0.64, 0.78, -1.10],
[ 1.78, 1.22, 1.12, -2.35],
[-0.48, -0.40, 1.73, 0.54],
[ 1.28, -0.18, 0.52, 2.10],
[ 0.34, 0.62, -0.45, -0.64]

[-0.22,-0.66,-1.00, -0.04],
[-0.23, -0.07, -0.28, 1.17],
[ 1.44, -0.64, 0.78, -1.10],
[ 1.78, 1.22, 1.12, -2.35],
[-0.48, -0.40, 1.73, 0.54],
[ 0.70, -1.35, 0.15, -1.44]
```

Output (3,6,4)

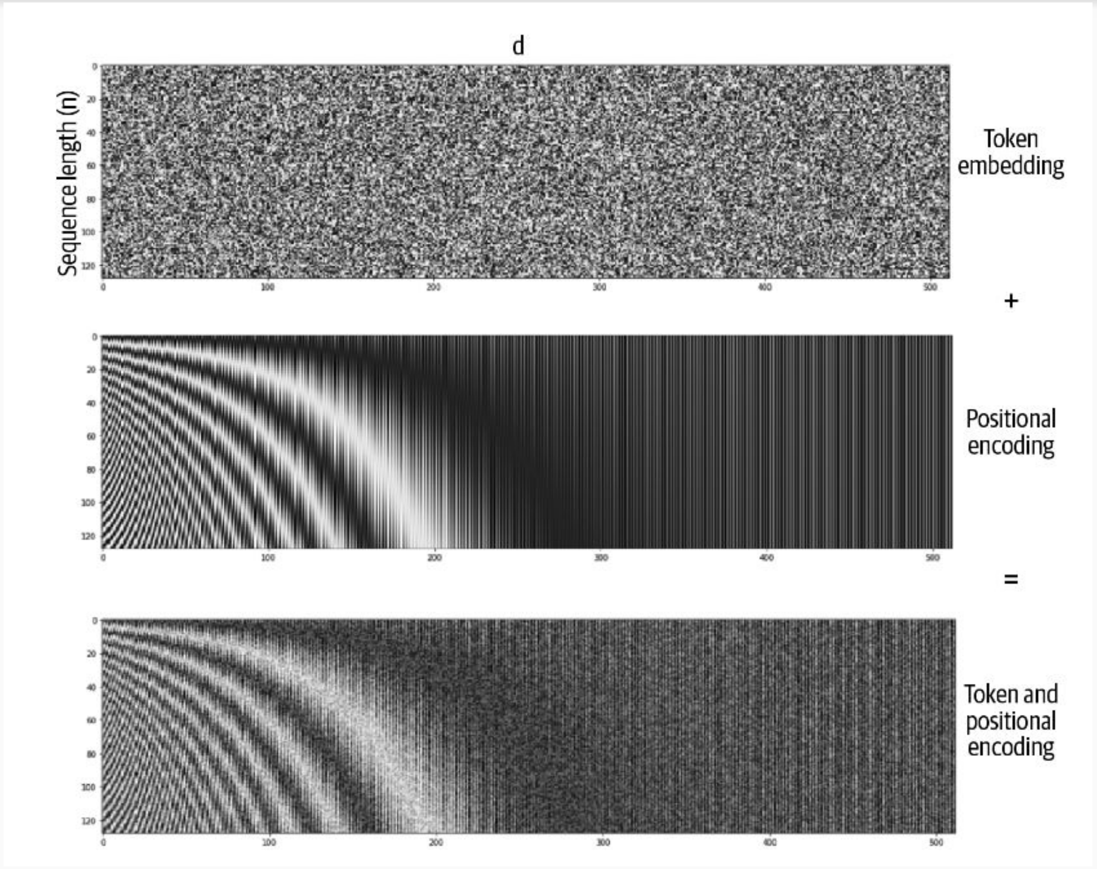
```
[-0.27, 0.18, 0.33, 2.39],
[ 2.57, -0.09, -1.03, 1.09],
[ 0.68, -0.49, -0.08, 2.15],
[ 0.75, 0.47, 1.50, 1.80],
[-2.80, 1.12, 1.90, 0.71],
[-0.10, -1.53, 1.03, 1.86]

[ 0.06, 0.66, 2.08, -0.24],
[ 2.28, -0.10, 0.88, -0.10],
[ 2.69, 0.80, 1.32, -1.37],
[-0.34, -1.39, 2.03, 1.50],
[ 0.52, -0.83, 0.91, 3.02],
[-0.62, 0.90, 0.03, 0.23]

[-0.22, 0.34, -1.00, 0.96],
[ 0.61, 0.47, -0.18, 2.16],
[ 2.35, -1.06, 0.98, -0.12],
[ 1.92, 0.23, 1.41, -1.40],
[-1.24, -1.06, 2.12, 1.46],
[-0.26, -1.06, 0.63, -0.56]
```

https://medium.com/@hunter-j-phillips/positional-encoding-7a93db4109e6

Token + Positional Encoding



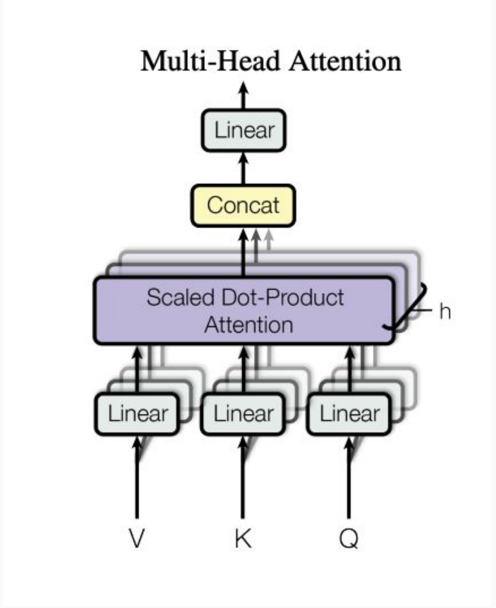
Foster, D. (2022). *Generative deep learning*. " O'Reilly Media, Inc.".

Multi-Headed Self-Attention

- Transformer models like GPT use multi-headed self-attention
- Self-attention: relays different input positions, modulated by context
- Allows model to jointly attend to information from different positions
- Each attention head focuses on different parts of context

Multihead Attention

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.
- Multihead attention concatenates the output from multiple attention heads, allowing each to learn a distinct attention mechanism so that the layer as a whole can learn more complex relationships.
- The concatenated outputs are passed through one final weights matrix W_o to project the vector into the desired output dimension.

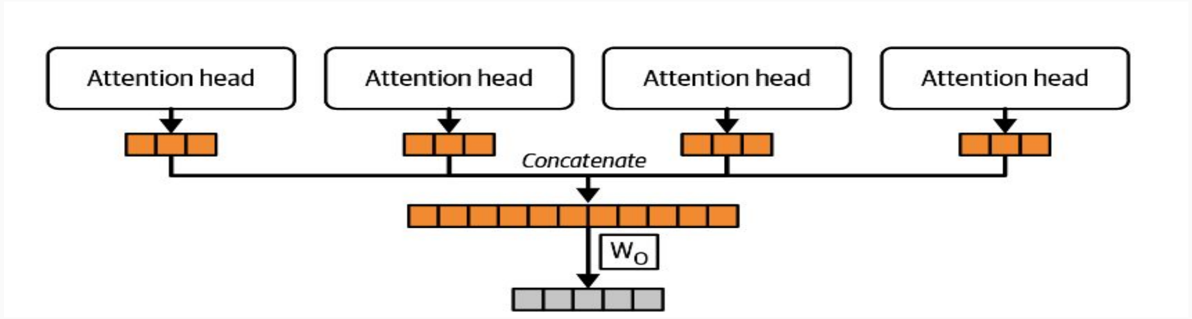


Foster, D. (2022). *Generative deep learning*. " O'Reilly Media, Inc.".

Multihead Attention

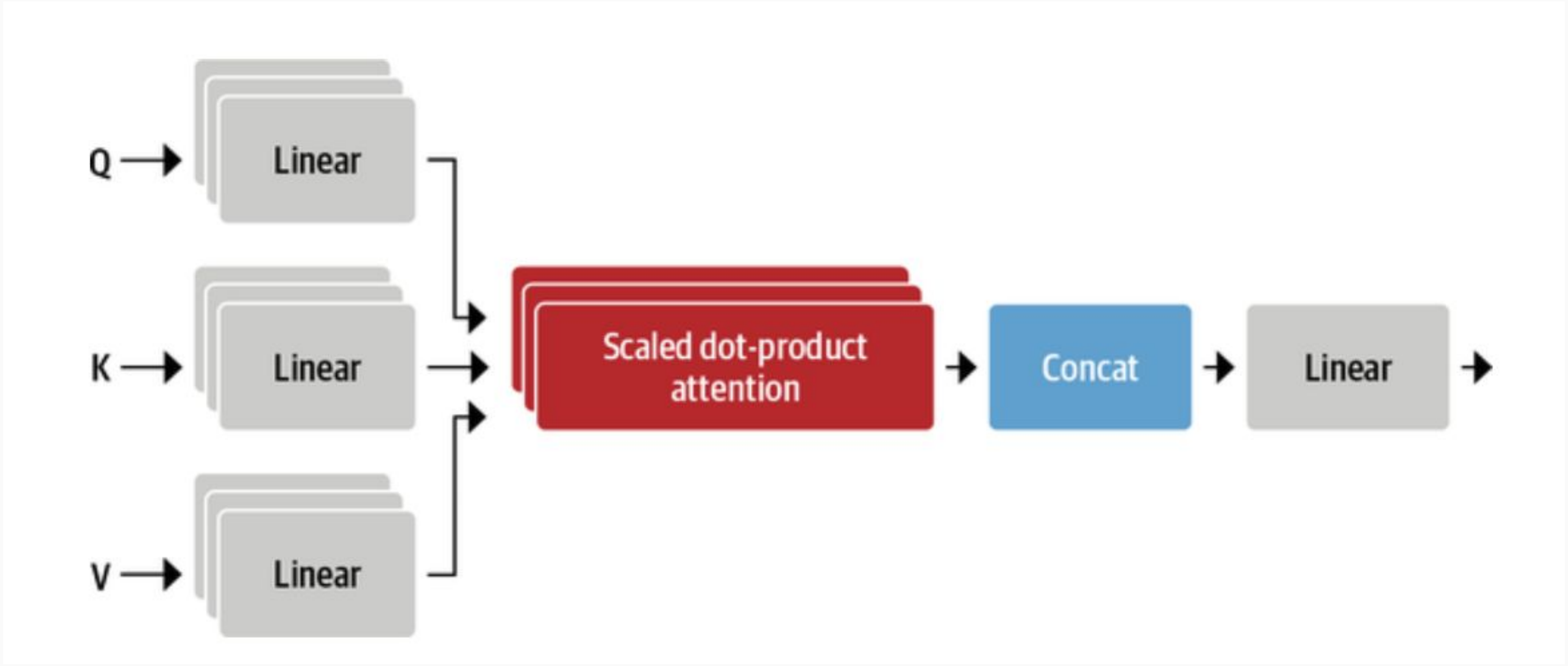
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.



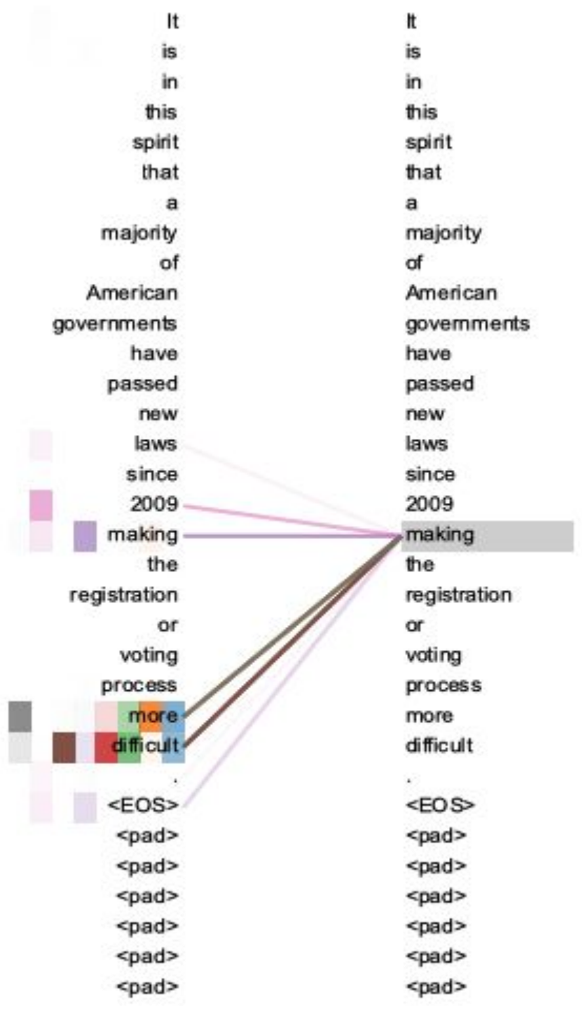
Foster, D. (2022). *Generative deep learning*. " O'Reilly Media, Inc.".

Recap - Multihead Attention



Example - Multi-Headed Self-Attention

- 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.

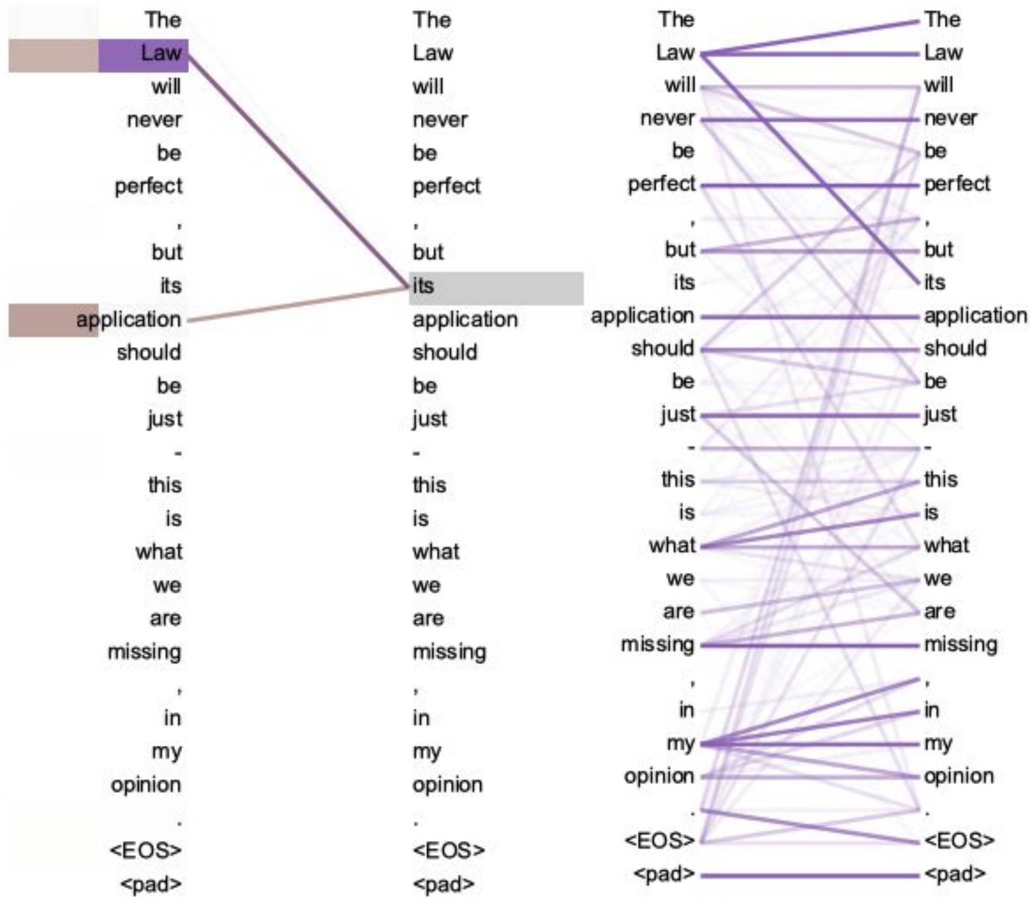


It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>
It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>

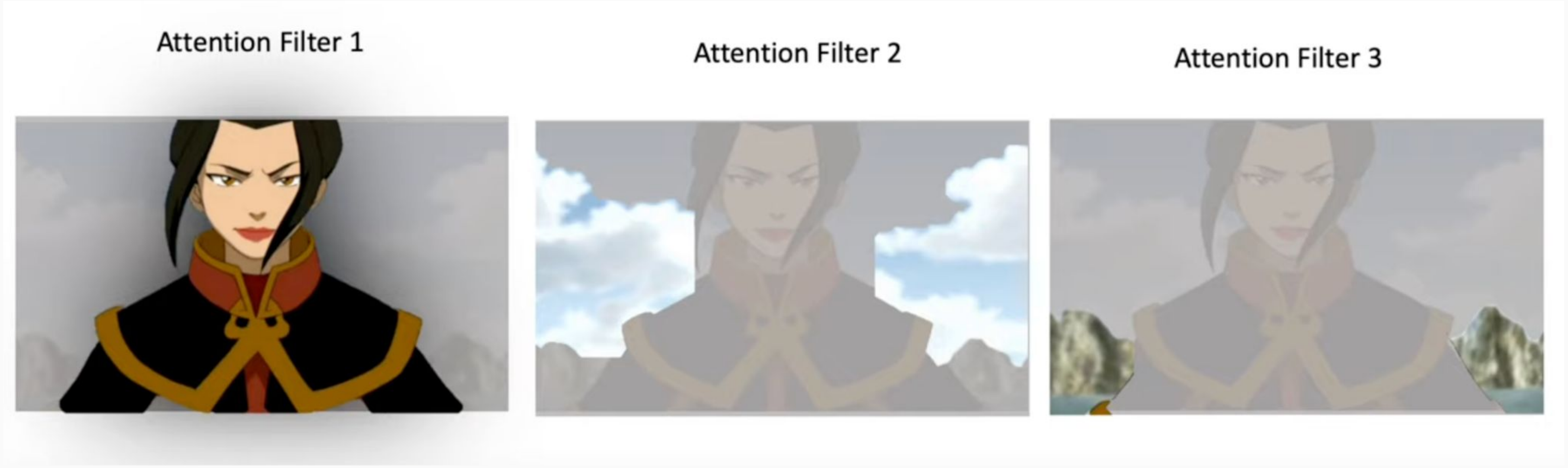
Anaphora resolution

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

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*.



Attention filters -> Multihead attention



Layernorm and Residual Connections

Layer Norm

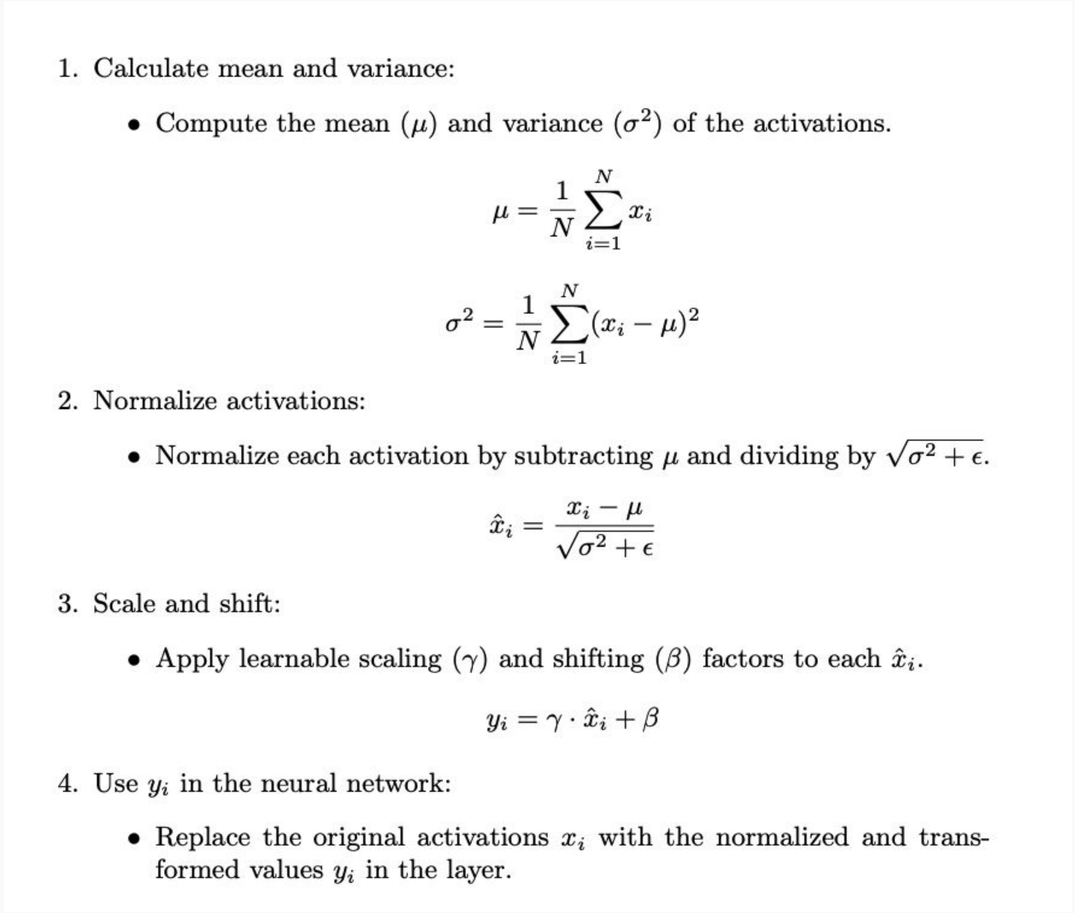
- Layer normalization is a technique proposed by Ba et al.(2016) to normalize the activations of neural network layers.
- It works by normalizing the outputs of a layer to have mean 0 and standard deviation 1 before they are fed into the next layer.
- This is done independently within each training case and for each element of the layer's output vector.
- In Transformers, layer normalization is applied on the input to each sub-layer, before it goes through functions like multi-headed attention and position-wise feed forward networks.

Why Layer Norm is needed?

- Applying layer normalization right before each sub-layer helps standardize the scale of the inputs. This makes it easier to learn stable representations.
- Layer normalization also helps regulate the scale of values flowing through different parts of the Transformer model.
- It stabilizes the hidden state dynamics in the self-attention layers across time-steps.
- Overall, layer normalization enables faster convergence of the Transformer during training by preventing instability.

Mechanics

In a neural network layer with *N* activations $x_1, x_2, \ldots, x_N$



1. Calculate mean and variance:
   - Compute the mean (μ) and variance (σ^2) of the activations.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

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

2. Normalize activations:
   - Normalize each activation by subtracting μ and dividing by $\sqrt{\sigma^2 + \epsilon}$.

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

3. Scale and shift:
   - Apply learnable scaling (γ) and shifting (β) factors to each $\hat{x}_i$.

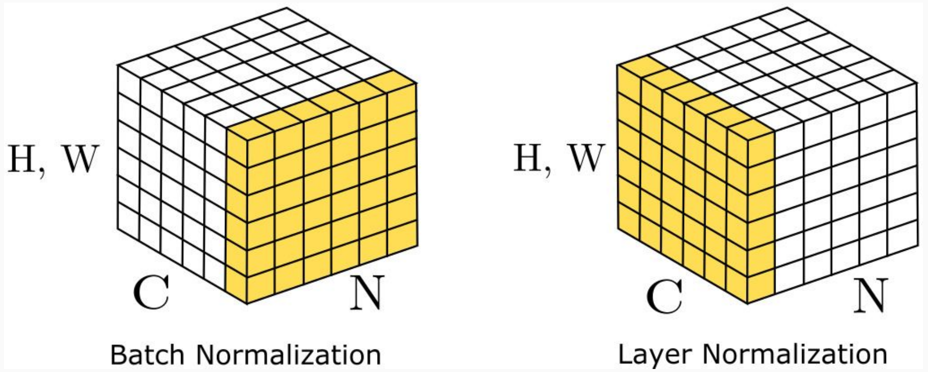
$$y_i = \gamma \cdot \hat{x}_i + \beta$$

4. Use y_i in the neural network:
   - Replace the original activations x_i with the normalized and transformed values y_i in the layer.

Implementation

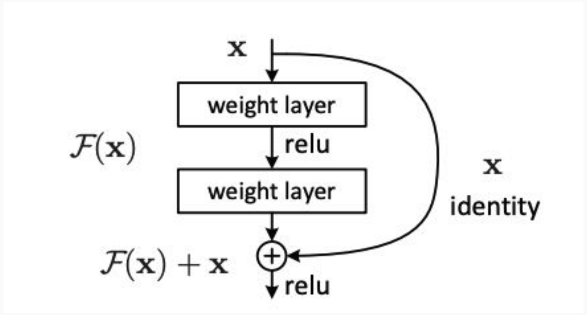
- Given a sample of shape [N, C, H, W]
- LayerNorm calculates a mean and variance of all the elements of shape [C, H, W] in each batch
- Implementing Layer Normalization
  - in PyTorch torch.nn.LayerNorm()
  - similarly in Keras keras.layers.LayerNormalization()

- Where,
  a. N is batch size - the number of samples in the input
  b. C stands for the number of channels (feature maps)
  c. H stands for the height of each channel
  d. W stands for the width of each channel



Residual Learning

- Deep networks suffer from degradation - increased training error with depth, even though representational capacity increases.
- Current optimization methods have trouble fitting identity mappings with stacked nonlinear layers.
- Identity mappings are necessary for preserving information flow through more layers.
- But solvers struggle to initialize weights to identity, and get stuck fitting random noise-like mappings.



He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Shortcut Connections

- Residual learning provides a new formulation to make optimization easier:
    - Don't have layers directly fit desired mapping H(x)
    - Instead fit residual mapping F(x) = H(x) - x
    - Original mapping becomes F(x) + x
- Identity function now built into mapping through shortcut connections.
- F=0 perfectly preserves identity mapping.
- Layers only need to fit residual F w.r.t. identity, not identity and H(x) directly.
- This reformulation eases difficulties in optimizing very deep representations.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Modeling Long-Range Dependencies

- Attention direct connections mitigate vanishing gradient problem with RNN backpropagation
- Enables learning dependencies even over longer distances

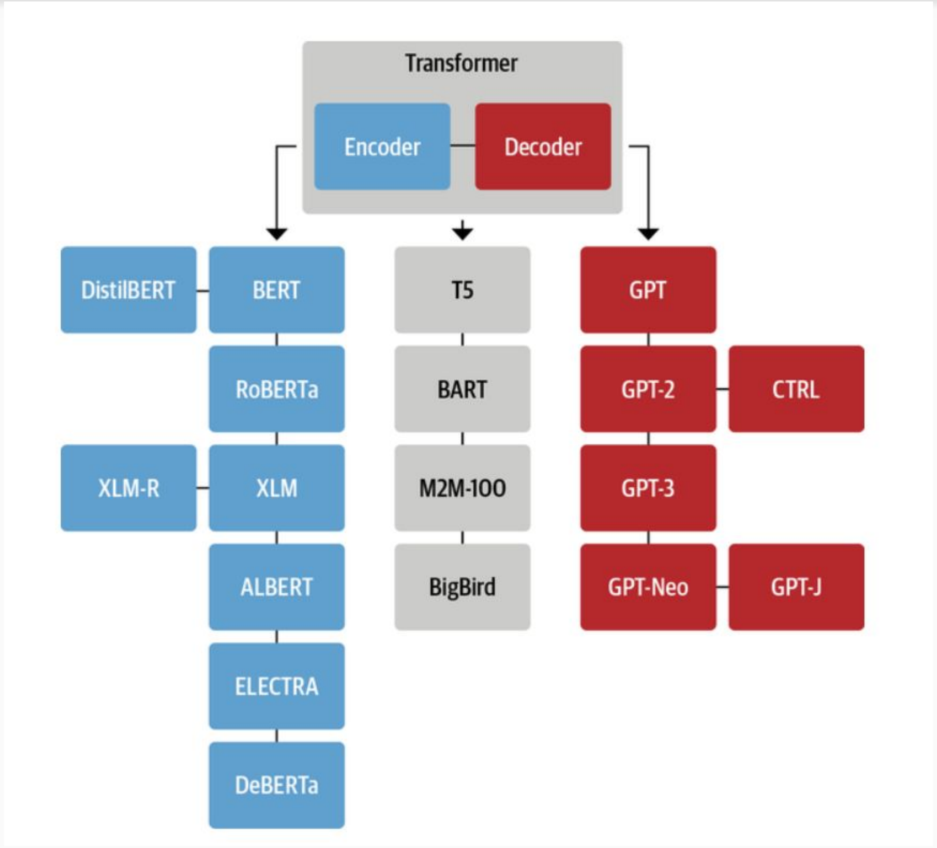
Why Attention is effective

- Attention filters input based on current context
- Selectively retrieves only relevant context for prediction
- Avoids retaining/encoding irrelevant past context

  -parameter-efficient: less parameters needed than RNN models

Transformer Architecture

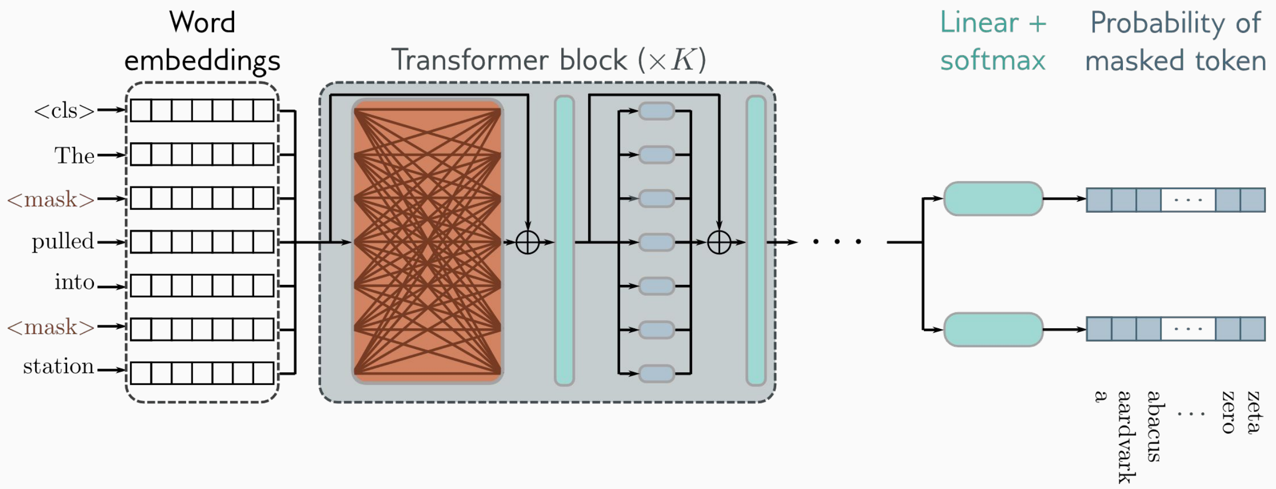
An overview of transformer architectures



Tunstall, L., Von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers*. " O'Reilly Media, Inc.".

Encoder Only

- These models convert an input sequence of text into a rich numerical representation that is well suited for tasks like text classification or named entity recognition.
- BERT and its variants, like RoBERTa and DistilBERT, belong to this class of architectures.
- The representation computed for a given token in this architecture depends both on the left (before the token) and the right (after the token) contexts.

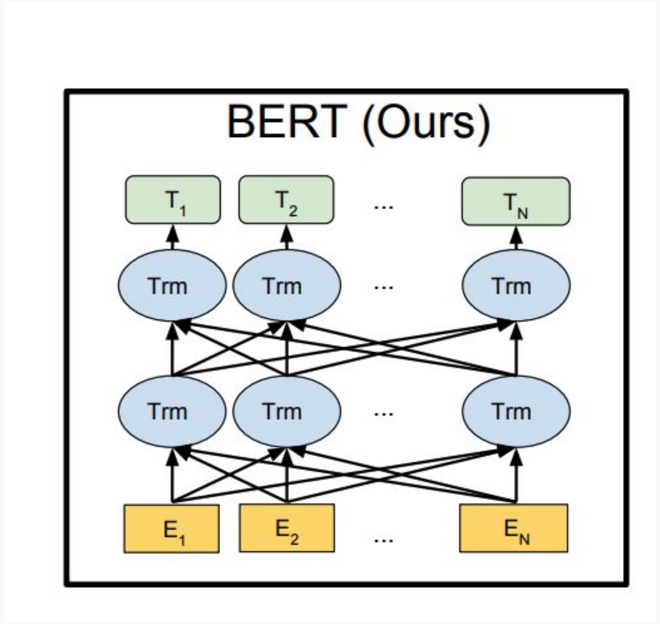


Tunstall, L., Von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers*. " O'Reilly Media, Inc.".

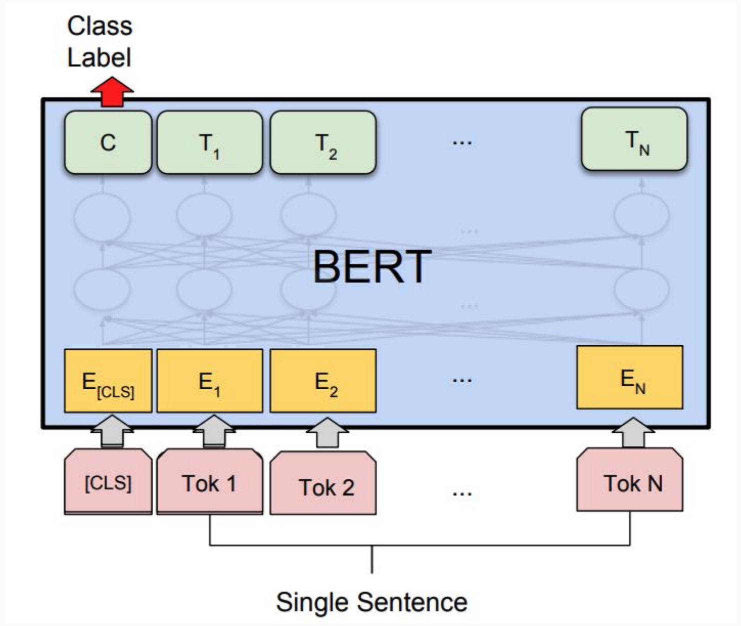
Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

BERT - Architecture

Generic Representation



Sentence Classification



Bidirectional Encoder Representations from Transformers (BERT)

- BERT is designed to pretrain deep bidirectional representations by jointly conditioning on both left and right context.
- This allows BERT to be easily fine-tuned to create state-of-the-art models for a wide range of NLP tasks without substantial task-specific modifications.
- BERT introduces two novel unsupervised pre-training tasks: Masked Language Modeling and Next Sentence Prediction.
- The model architecture is a multi-layer bidirectional Transformer encoder.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*

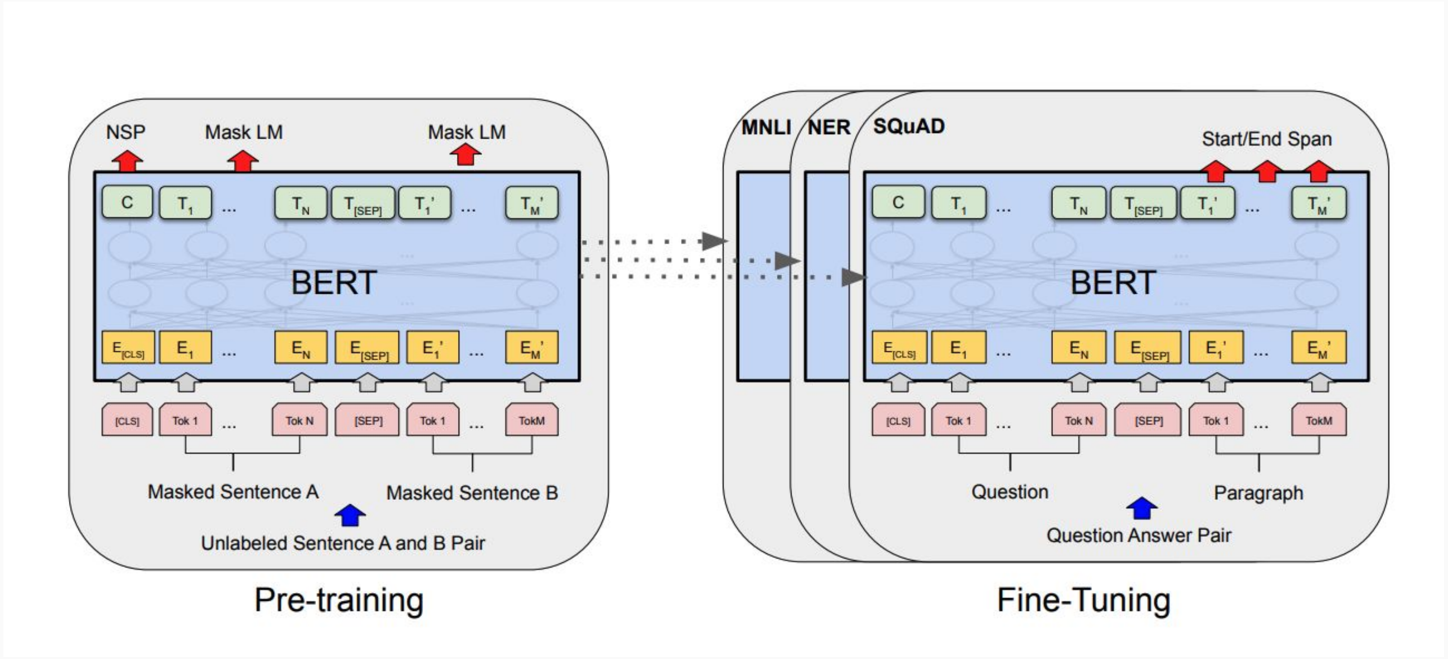
Pre-training and Fine tuning

Pre-training a language model refers to first training a model on a large general-domain corpus of data before fine-tuning it on a downstream task.

- Objective is to teach the model about general properties and regularities of language by learning to predict words/tokens based on the context. Typically done on large unlabeled corpora like BooksCorpus, Wikipedia etc. that provide broad coverage of language.
- Helps the model learn effective word representations and capture semantics, syntax, grammar etc.Often uses a standard language modeling training objective of predicting next token given context.
- Adds inductive bias to the model by pre-loading linguistic knowledge, allowing it to generalize better for downstream tasks, especially with small data.
- Results in a general-purpose model that captures universal language information.

This model is then fine-tuned on downstream tasks.

BERT - Pre-training and Fine tuning



- Same architectures are used in both pre-training and fine-tuning.
- The same pre-trained model parameters are used to initialize models for different downstream tasks.

Pre-training Data

- BooksCorpus (800M words) (Zhu et al., 2015)
- and English Wikipedia (2,500M words).

Model Size

- $\text{BERT}_{\text{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M)
- $\text{BERT}_{\text{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M).

Number of layers (i.e., Transformer blocks) is L, the hidden size is H, and the number of self-attention heads are A

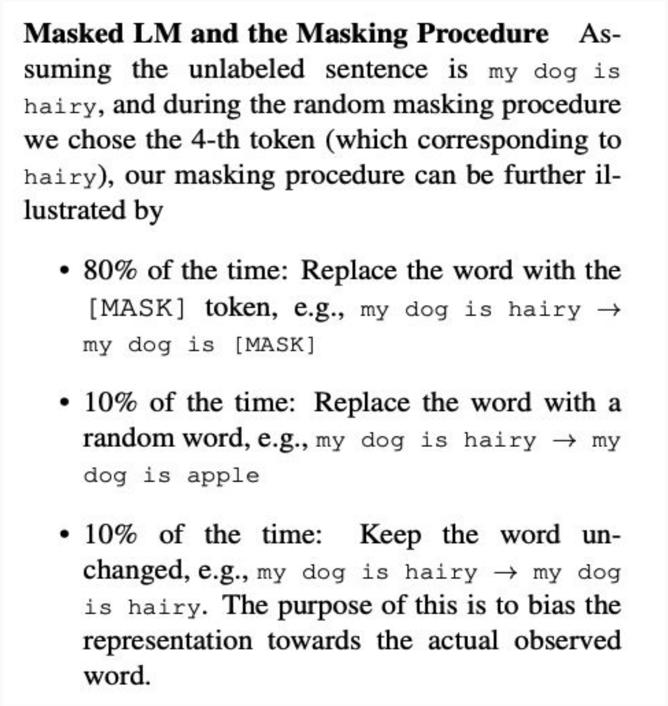
Fine-tuning data

- Stanford Question Answering Dataset (SQuAD) is a collection of 100k crowdsourced question/answer pairs (Rajpurkar et al., 2016).
- Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples that evaluate grounded common sense inference (Zellers et al., 2018).
- MRPC Microsoft Research Paraphrase Corpus consists of sentence pairs automatically extracted from online news sources, with human annotations (Dolan and Brockett, 2005).

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*

Pre-training: Masked Language Model (MLM)

- The masked language model (Taylor, 1953) randomly masks some of the tokens from the input.
- The objective is to predict the original vocabulary id of the word based only on its context.
- Fuse the left and the right context, which allows us to pretrain a deep bidirectional Transformer.

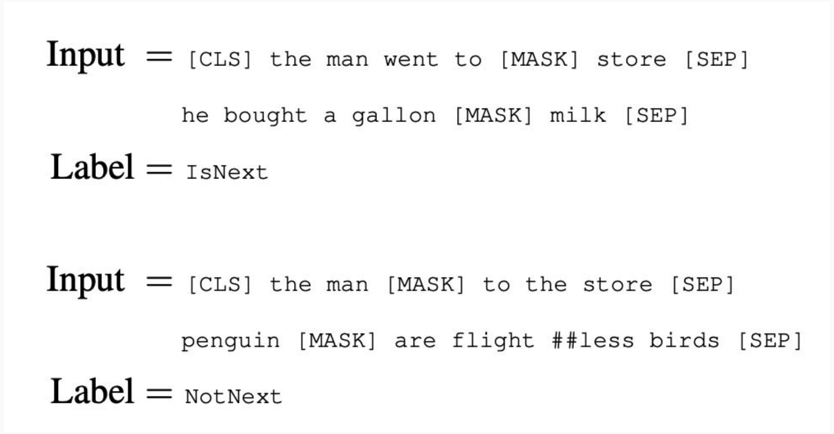


Masked LM and the Masking Procedure Assuming the unlabeled sentence is `my dog is hairy`, and during the random masking procedure we chose the 4-th token (which corresponding to `hairy`), our masking procedure can be further illustrated by

- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

Pre-training: Next Sentence Prediction (NSP)

- Used to capture the relationship between two sentences
- Binarized next sentence prediction
    - When choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext)
    - and 50% of the time it is a random sentence from the corpus (labeled as NotNext).

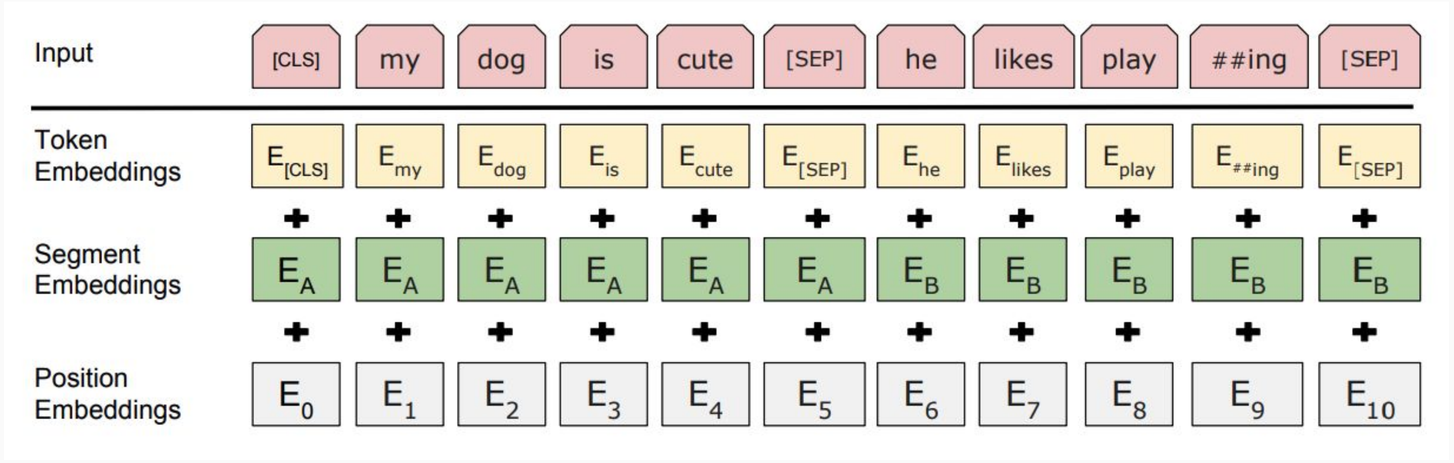


Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

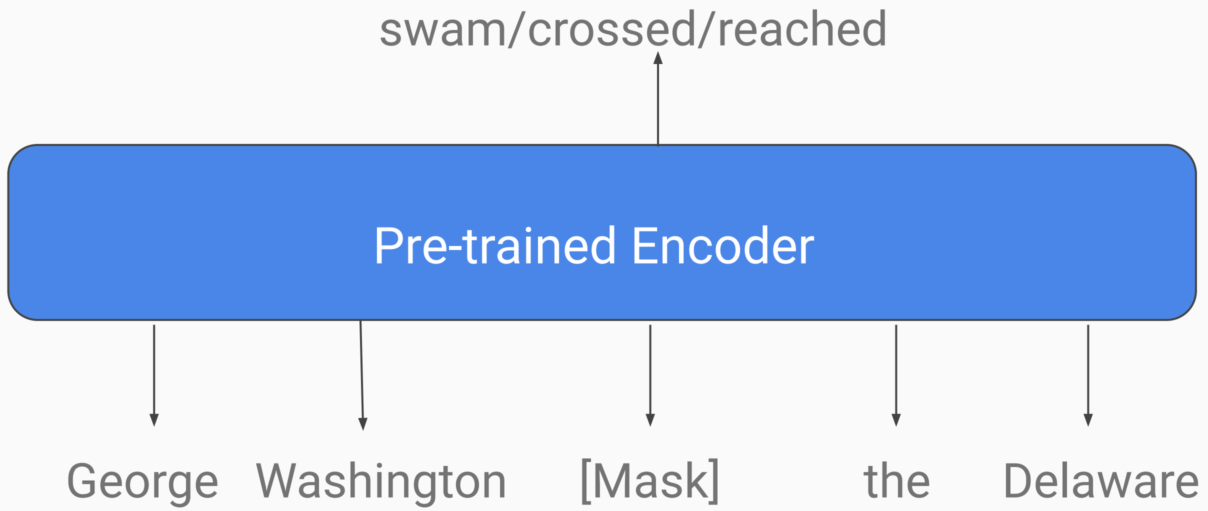
BERT Input representation

Input embeddings = Token embeddings + Segmentation embeddings + Position embeddings.


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

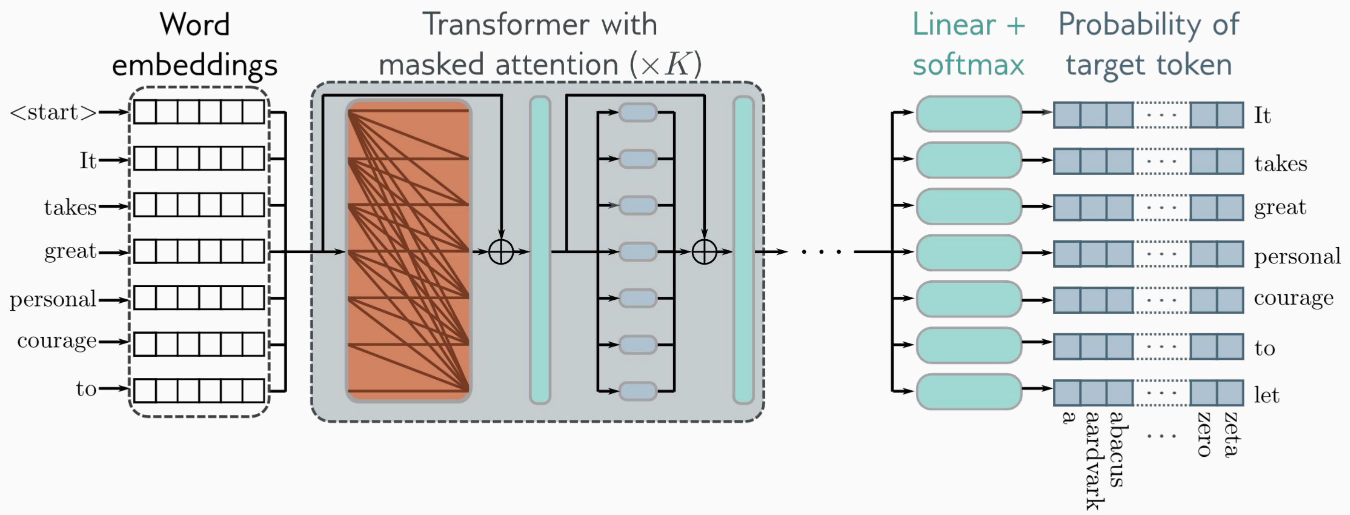
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*

Limitations of Pre-trained Encoder



Decoder Only (*causal* or *autoregressive attention*)

- Will auto-complete the sequence by iteratively predicting the most probable next word.
- The representation computed for a given token in this architecture depends only on the left context i.e. the output of the decoder is NOT conditioned on future words.
- With just a decoder module and no encoder, a decoder-only architecture uses approximately half the parameters of common Encoder-Decoder models.



Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

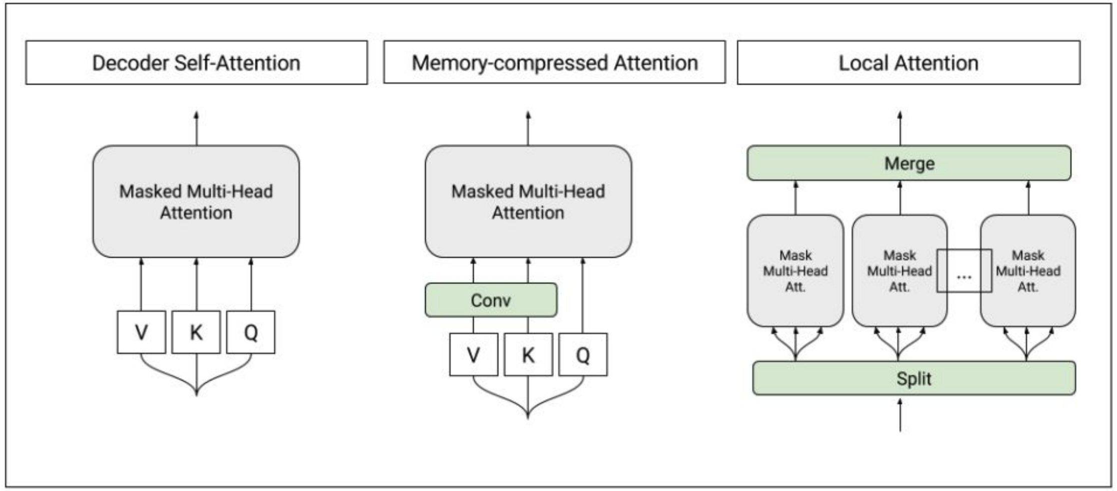
Transformer Decoder With Memory-Compressed Attention (T-DMCA)

The multi-head self-attention is modified to reduce memory usage by limiting the dot products between Q and K.

- Local attention: Sequence tokens are divided into blocks of similar length and attention is performed in each block independently.
- As the attention memory cost per block becomes constant, this modification allow us to keep the number of activations linear with respect to the sequence length.
- Memory-compressed attention: Reduce the number of keys and values using a strided convolution.
  - The number of queries remains unchanged.
  - This modification allows the Transformer to divide the number of activations.
  - Convolution kernels are of size 3 with stride 3.
  - The local attention layers capture the local information within a block. However, the memory compressed attention layers are able to exchange information globally on the entire sequence.

P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. ICLR, 2018.

T-DMCA - architecture



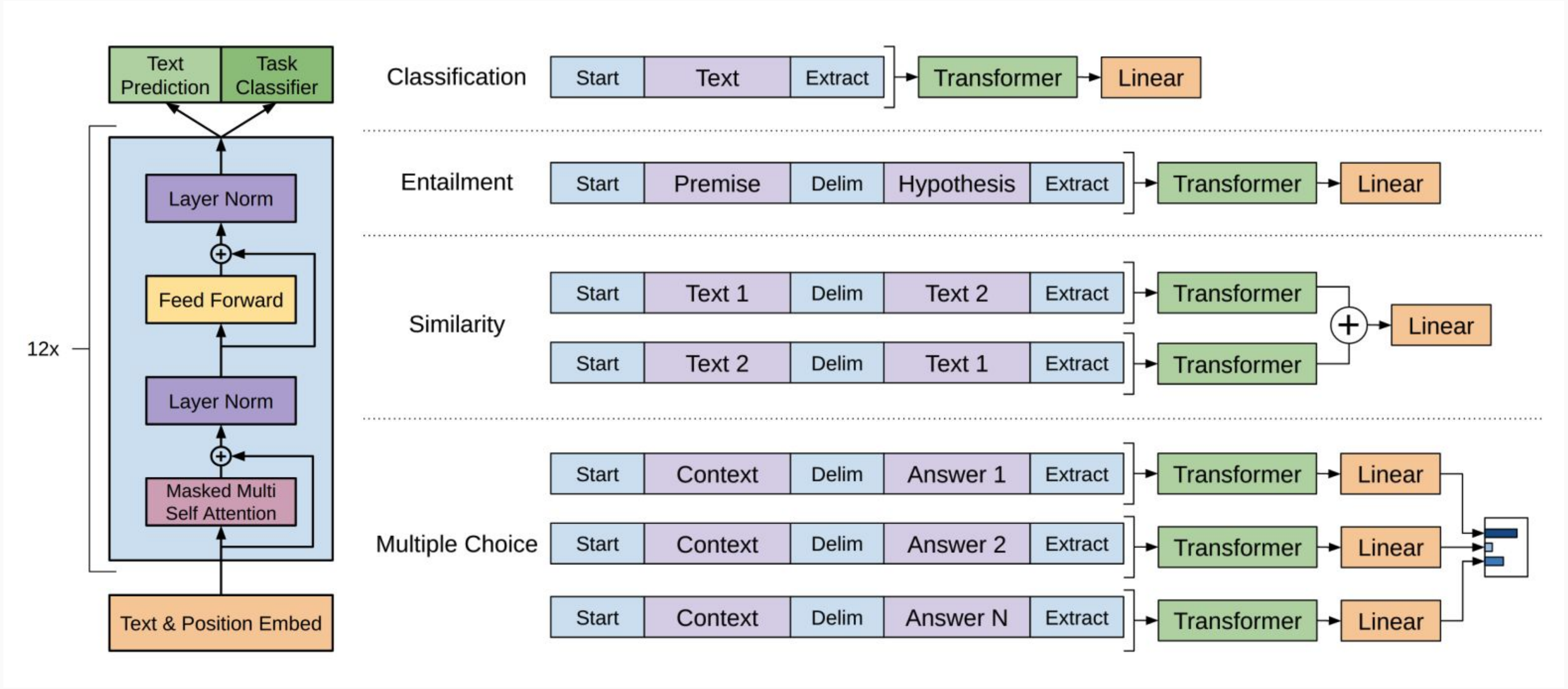
- Left: Original self-attention as used in the transformer-decoder.
- Middle: Memory-compressed attention which reduce the number of keys/values.
- Right: Local attention which splits the sequence into individual smaller sub-sequences. The sub-sequences are then merged together to get the final output sequence.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., & Shazeer, N. (2018). Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*.

Generative Pre-Training (GPT)

- Semi-supervised approach for natural language understanding tasks using unsupervised pre-training of a Transformer language model on a large corpus of text, followed by supervised fine-tuning on specific target tasks.
- The pre-trained model captures long-range linguistic structure and world knowledge that transfers effectively to downstream tasks with minimal architectural changes, only requiring task-specific input adaptations and an output layer.

GPT - Architecture

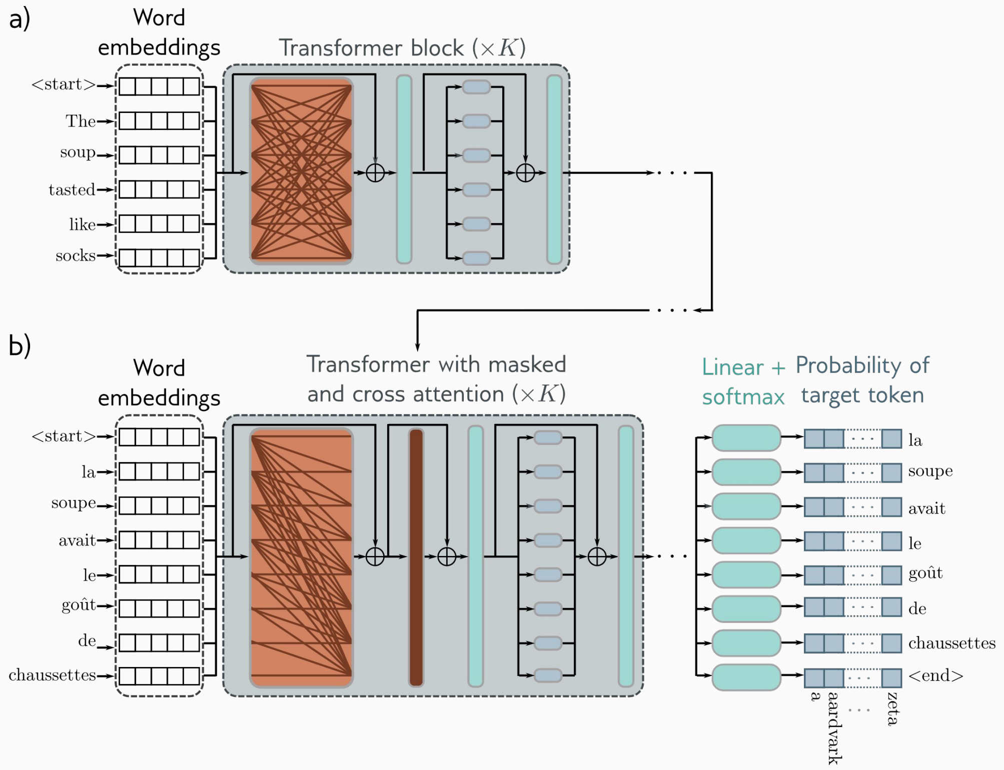


(left) Transformer architecture, (right) Input transformations for fine-tuning on different tasks

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

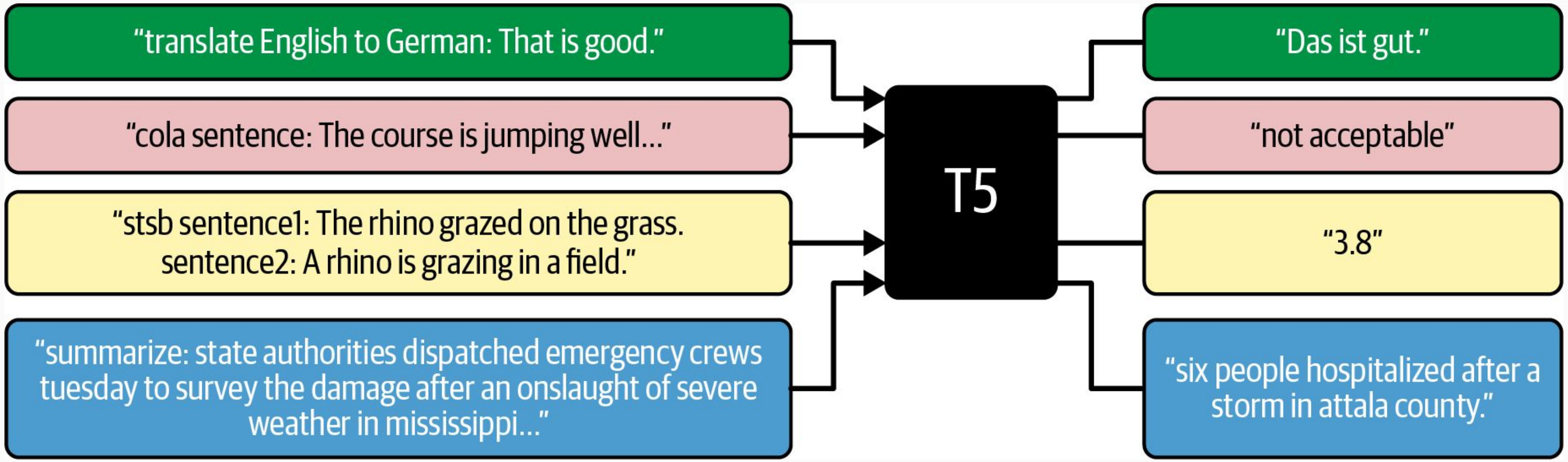
Encoder-Decoder

- Used for modeling complex mappings from one sequence of text to another.
- They're suitable for machine translation and summarization tasks.
- BART and T5 models belong to this class.



Text-to-Text Transfer Transformer (T5)

Every task - translation, question answering, and classification—is cast as feeding the model text as input and training it to generate some target text.


"translate English to German: That is good."
"cola sentence: The course is jumping well..."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi..."
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, *21*(1), 5485-5551.

Encoder

- Composed of a series of transformer blocks
- Self-attention layers in the blocks have no masking, allowing each word to attend to all other words irrespective of position.
- This lets the encoder build deep interconnected representations capturing dependencies across the entire input sequence.

Decoder

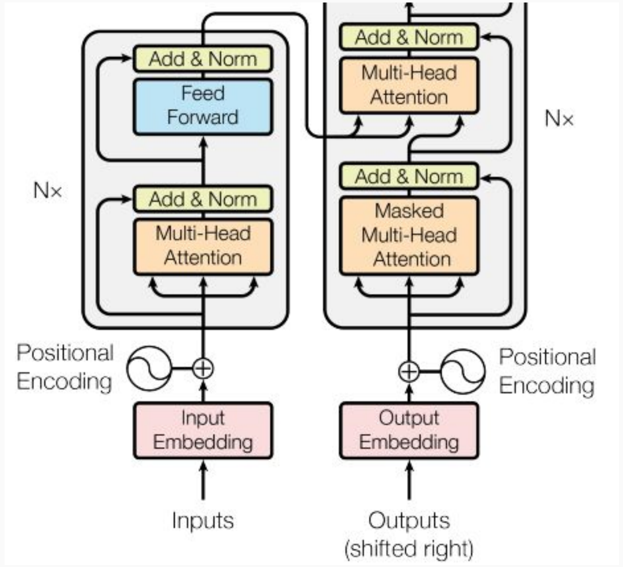
- Composed of a series of transformer blocks
- Uses two types of attention layers:
    1. Causal Self-Attention:
        - Self-attends over decoded tokens generated so far
        - Uses causal masking so no information about future tokens leaks to current token
    2. Cross-Attention to Encoder:
        - Attends to encoder representations
        - Allows deciding what meaning needs to be translated based on the input sequence's encodings

The need for cross-attention

- In encoder-decoder transformers information needs to flow from the encoder to the decoder to relate the input and output sequences.
- Self-attention layers allow the encoder to build representations capturing relationships within the input sequence.
- The decoder needs to leverage these encoder representations to determine appropriate output tokens.
- This is achieved via cross-attention layers in the decoder.
- Cross-attention performs regular dot-product attention.
- However, instead of comparing a token to other tokens from the same sequence, it compares a decoder token to all tokens from the encoder sequence.

Cross-attention

- The queries come from the previous decoder layer, and the keys and values come from the output of the encoder.
- This allows each decoder token to determine which encoded input tokens are most relevant to the next predicted output.
- The attention scores determine which elements of the input sequence are most pertinent to the current decoding step.
- The context vector obtained from weighted averaging using the attention weights helps predict the next decoder token accordingly.



Unidirectional autoregressive behavior

- Autoregressive models generate sequences one token at a time in a unidirectional left-to-right manner
- The probability distribution of the next token only conditions on the tokens that have already been predicted earlier
- If future predicted tokens could influence past tokens, it would violate the autoregressive property
- This conditional dependence of future only on the past is crucial for autoregressive generation

Masking - restricts information flow to only the leftward direction

- Self-attention layers allow attention to all positions by default
- Masking prevents each position in the decoder from attending to future positions on the right
- Masking enforces the unidirectional autoregressive behavior needed for decoding
- Without masking, the model could peek at future tokens that should not inform current prediction.
- At every timestep, we could change the set of keys and queries to include only past words. (Inefficient!)
- To enable parallelization - mask out attention to future words by setting attention scores to $-\infty$.

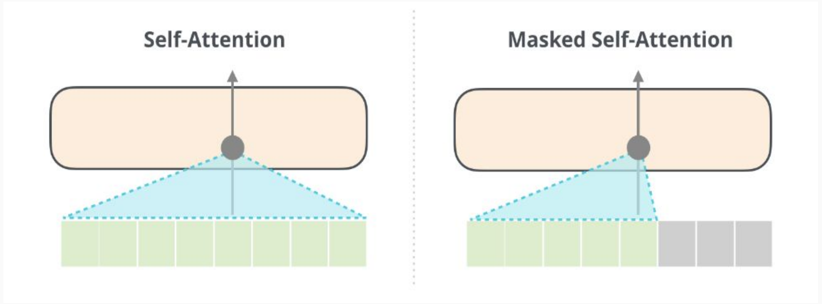


Image from https://jalammar.github.io/illustrated-gpt2/

Implementing Masking

$$QK^T = \begin{array}{c|cccc} & a^K & b^K & c^K & D^K \\ \hline a^Q & (a^Q a^K & a^Q b^K & a^Q c^K & a^Q D^K)_s \\ b^Q & (b^Q a^K & b^Q b^K & b^Q c^K & b^Q D^K)_s \\ c^Q & (c^Q a^K & c^Q b^K & c^Q c^K & c^Q D^K)_s \\ D^Q & (D^Q a^K & D^Q b^K & D^Q c^K & D^Q D^K)_s \end{array}$$

$$M = \begin{array}{c|cccc} & a^K & b^K & c^K & D^K \\ \hline a^Q & 0 & -\infty & -\infty & -\infty \\ b^Q & 0 & 0 & -\infty & -\infty \\ c^Q & 0 & 0 & 0 & -\infty \\ D^Q & 0 & 0 & 0 & 0 \end{array}$$

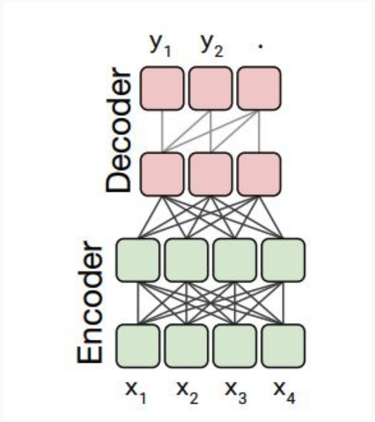
$$Soft(QK^T + M) = \begin{array}{c|cccc} & a^K & b^K & c^K & D^K \\ \hline a^Q & (a^Q a^K & -\infty & -\infty & -\infty)_S \\ b^Q & (b^Q a^K & b^Q b^K & -\infty & -\infty)_S \\ c^Q & (c^Q a^K & c^Q b^K & c^Q c^K & -\infty)_S \\ D^Q & (D^Q a^K & D^Q b^K & D^Q c^K & D^Q D^K)_S \end{array}$$

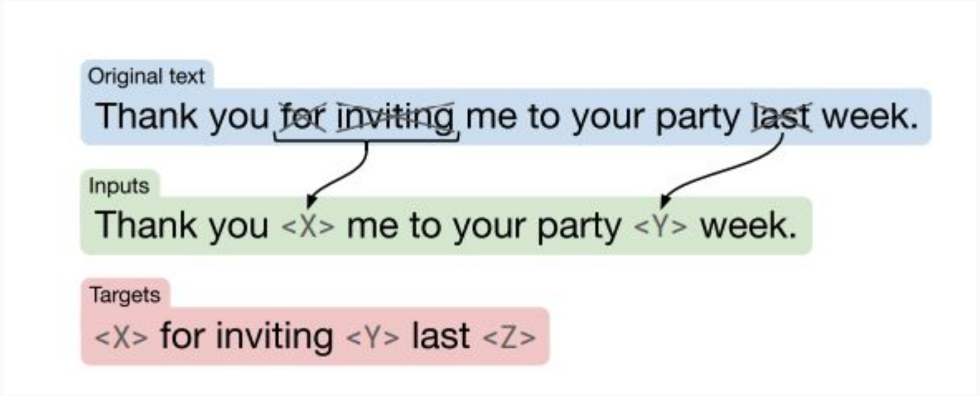
$$= \begin{array}{c|cccc} & a^K & b^K & c^K & D^K \\ \hline a^Q & a^Q a^K & 0 & 0 & 0 \\ b^Q & (b^Q a^K & b^Q b^K)_S & 0 & 0 \\ c^Q & (c^Q a^K & c^Q b^K & c^Q c^K)_S & 0 \\ D^Q & (D^Q a^K & D^Q b^K & D^Q c^K & D^Q D^K)_S \end{array}$$

https://medium.com/mlearning-ai/how-do-self-attention-masks-work-72ed9382510f

Autoregression

- The decoder generates translations autoregressively using causal self-attention over previously predicted tokens.
- Cross-attention over the context-rich encoder representations allows determining appropriate translated text that conveys the meaning of the input sequence.





Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Reality is a bit blurry

- Decoder-only models like those in the GPT family can be primed for tasks like translation that are conventionally thought of as sequence-to-sequence tasks.
- Similarly, encoder-only models like BERT can be applied to summarization tasks that are usually associated with encoder-decoder or decoder-only models.

Tunstall, L., Von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers*. " O'Reilly Media, Inc.".

Causal mediation analysis

Causal mediation analysis

- Causal mediation analysis is a statistical method used to understand the mechanisms through which an independent variable can affect a dependent variable through one or more mediator variables.
- This analysis is crucial in many fields, including psychology, economics, and increasingly in machine learning, especially in understanding complex models like neural networks.
- In the context of neural networks, particularly language models like GPT, causal mediation analysis helps to identify which components (e.g., specific neurons or layers) mediate the relationship between the input (like a sentence or phrase) and the model's output (like the prediction of the next word or a factual answer).

Core Concepts of Causal Mediation Analysis

1. Independent Variable (IV): This is the variable that is presumed to cause an effect on the dependent variable. In neural networks, this could be the input to the model.
2. Dependent Variable (DV): This is the outcome of interest, which in the context of neural networks, might be the prediction accuracy or the output of the model.
3. Mediator Variable: These are the internal states or activations within the model that potentially mediate the effect of the IV on the DV. In language models, these could be the activations in specific layers or neurons.
4. Direct Effect: This is the effect of the independent variable on the dependent variable that is not mediated by the mediator. It represents the relationship that bypasses the mediator.
5. Indirect Effect: This is the effect of the independent variable on the dependent variable that passes through the mediator. It quantifies how much of the total effect is mediated through changes in the mediator.

Steps in Causal Mediation Analysis

1. Model Setup: Define and setup the model with the independent variable, mediator, and dependent variable clearly specified. For a neural network, this involves identifying the input variables, internal layers or neurons as mediators, and the output variables.
2. Estimation of Effects: Use statistical or computational methods to estimate the direct and indirect effects. In machine learning:
   - The total effect is often estimated by observing the change in the output when the input is varied.
   - The direct effect is estimated by controlling for the mediator, essentially setting the mediator to a fixed value and varying the input to see the effect on the output.
   - The indirect effect is estimated by observing how changes in the input affect the mediators, and in turn, how these changes in the mediators affect the output.
3. Intervention: Perform interventions on the mediators by artificially manipulating their states (e.g., by "clamping" a neuron's output) and observing the resultant effect on the output. This helps to confirm the role of these mediators.
4. Statistical Testing: Apply statistical tests to confirm the significance of the observed direct and indirect effects. This often involves comparing the model's predictions with and without the intervention to see if the changes are statistically significant.

Application in Neural Networks

- In neural networks, causal mediation analysis can be particularly challenging due to the high dimensionality and the complex, non-linear interactions between components.
- However, it provides valuable insights into how information is processed within the network and how different parts of the network contribute to the final decision-making process.
- By understanding these pathways, researchers and engineers can design better and more interpretable models
- Potentially intervene in these models to correct or alter their behavior in desired ways
- For example, modifying a language model to update or correct its stored knowledge based on new information.

Transformers and Memory

Feed-forward layers in transformer models

- What kind of memories are stored in these feed-forward layers?
- What patterns do the keys capture and what is represented in their corresponding values?
- How does the final transformer output aggregate information from multiple feed-forward layer memories across layers?

Geva, M., Schuster, R., Berant, J., & Levy, O. (2020). Transformer feed-forward layers are key-value memories. arXiv preprint arXiv:2012.14913

What kind of memories are stored in these feed-forward layers?

The feed-forward layers in transformer models store memories as key-value pairs:

- Keys: Detect specific input patterns ranging from simple syntactic n-grams to complex semantic themes.
- Values: Each value represents a probability distribution over the output vocabulary, predicting likely next tokens based on the input pattern identified by the key.
- Functionality Across Layers:
    - Lower Layers: Store simpler patterns, focusing on basic textual elements like common suffixes.
    - Upper Layers: Handle more abstract and complex patterns, capturing semantic relationships and thematic contexts.

These layers essentially learn and remember associations between input patterns and their logical continuations, allowing the model to generate contextually relevant text based on these learned associations.

Geva, M., Schuster, R., Berant, J., & Levy, O. (2020). Transformer feed-forward layers are key-value memories. arXiv preprint arXiv:2012.14913

Example

- If a key is triggered by the input phrase "interest rates"
- The corresponding value might predict words like "increase", "decrease", or specific terms related to economic contexts.
- This shows that the feed-forward layer has learned to associate the phrase "interest rates" with a particular set of likely continuations, effectively storing this association as a memory.

Geva, M., Schuster, R., Berant, J., & Levy, O. (2020). Transformer feed-forward layers are key-value memories. arXiv preprint arXiv:2012.14913

What patterns do the keys capture and what is represented in their corresponding values?

- In transformer models, the keys within feed-forward layers capture input patterns that are either syntactic (e.g., specific word sequences) or semantic (e.g., thematic contexts).
- The values corresponding to these keys represent probability distributions over the output vocabulary, predicting the most likely subsequent tokens based on the detected patterns:
  - Syntactic Patterns: Detected by keys, could include grammatical structures like "the adj noun."
  - Semantic Patterns: Involve broader themes or actions, recognized by keys reacting to inputs like "canceling the meeting."
- The values then predict contextually appropriate continuations, ensuring the text generated by the model is coherent and contextually relevant.

Geva, M., Schuster, R., Berant, J., & Levy, O. (2020). Transformer feed-forward layers are key-value memories. arXiv preprint arXiv:2012.14913

Example

If a key is activated by the input "despite the rain," the corresponding value might predict words like "we," "they," or "the event" as likely continuations, shaping a sentence that logically flows from the given condition.

How does the final transformer output aggregate information from multiple feed-forward layer memories across layers?

The final output is produced by aggregating and refining information across multiple feed-forward layers, using the following steps:

1. Memory Composition: Each layer generates outputs based on key-value pairs that determine possible next tokens, influenced by input patterns.
2. Residual Connections: Outputs from each layer are added to outputs from previous layers via residual connections, helping to preserve information throughout the network.
3. Normalization and Activation: The combined output is normalized and passed through non-linear activations to stabilize learning and introduce necessary complexities.
4. Layer-by-Layer Refinement: As data progresses through successive layers, each layer refines earlier predictions, enhancing context understanding and adjusting probability distributions for next tokens.
5. Final Output Generation: The aggregated and refined data from the last layer undergoes a final transformation to determine the most probable next token, culminating in the model’s output.

This process ensures that transformer outputs are contextually informed and semantically accurate, leveraging the model's entire depth.

Geva, M., Schuster, R., Berant, J., & Levy, O. (2020). Transformer feed-forward layers are key-value memories. arXiv preprint arXiv:2012.14913

Memory Editing in Transformers

Why do we need *knowledge-editing* methods?

Michael Jordan plays the sport...
the model will predict *"basketball"*, a word that not only is grammatically correct, but that it is also consistent with a true fact in the real world.

However, the knowledge contained in a large language model is not perfect: even the largest models will be missing specialized knowledge, and a model will also contain obsolete knowledge that it learned from old text.

GPT-3 predicts: *Polaris is in the constellation Ursa Minor* (correct!)

GPT-3 predicts: *Arneb is in the constellation of Aquila* (incorrect - should be Lepus)

GPT-3 predicts: *The current Vice President of the United States is named Mike Pence* (obsolete)

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*.

Memory and fresh information

- Scope of Memory Recall - Large autoregressive language models can recall common facts efficiently but struggle with specialized knowledge and may retain outdated information.
- Need for Fresh Information - There's a demand for models that can dynamically update information across various applications like search engines and content generation.
- Challenges with Re-training - Fully retraining large models to update knowledge is resource-intensive and often impractical.

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*.

Existing Methods

- Knowledge bases
- Direct model editing
- Hypernetwork knowledge editing.
- Issues:
  - limited to small-scale updates.
  - focused on updating a single memory in the model
  - and it has been a challenge to use those methods to update more than a handful of facts.
- In practice we may want to insert hundreds or thousands of new memories in order to update or improve a large model.

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*.

Language Modeling and Memory Editing

1. LLMs Functionality: These models generate text by predicting one word at a time based on previous words, using complex internal mechanisms.
2. Memory in LLMs: LLMs store factual information, like "Michael Jordan plays basketball," within their parameters as patterns.
3. MEMIT's Objective: MEMIT updates these memorized facts, such as changing "Michael Jordan plays basketball" to "Michael Jordan plays baseball," and aims to do this for many facts simultaneously.
4. Editing Challenges:
   a. Effectiveness: The model should prioritize new facts over old ones.
   b. Generalization: The model should recall the new fact correctly in various phrasings.
   c. Specificity: Updates should not affect unrelated facts.
   d. Fluency: Text generated post-edit should remain natural and coherent.

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*.

Introduction of MEMIT

- Proposes a scalable solution for inserting thousands of memories directly into model weights, aiming to overcome the limitations of existing methods.
- By modifying transformer weights, MEMIT updates factual memories while maintaining generalization, specificity, and fluency, significantly outperforming other methods in large-scale memory updates.
- Tested on models like GPT-J and GPT-NeoX, MEMIT has proven effective for large-scale, robust knowledge updates.

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*.

Methodology

1. Identifying Key Layers: MEMIT uses causal analysis to find which layers in the model are critical for remembering specific facts.
2. Updating Memories: It then updates these layers by adjusting the data vectors that represent stored facts, focusing on the last word or subject in a sentence.
3. Batch Updates: Instead of updating memories one by one, MEMIT applies changes in batches to multiple memories at once, improving efficiency and accuracy.
4. Technical Approach: Through calculations, MEMIT ensures that new memories integrate well with existing ones without causing inaccuracies.

In short, MEMIT fine-tunes AI models to update and manage stored factual information effectively, focusing on crucial model layers and employing batch updates to optimize the process.

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*.

Critical Path Update

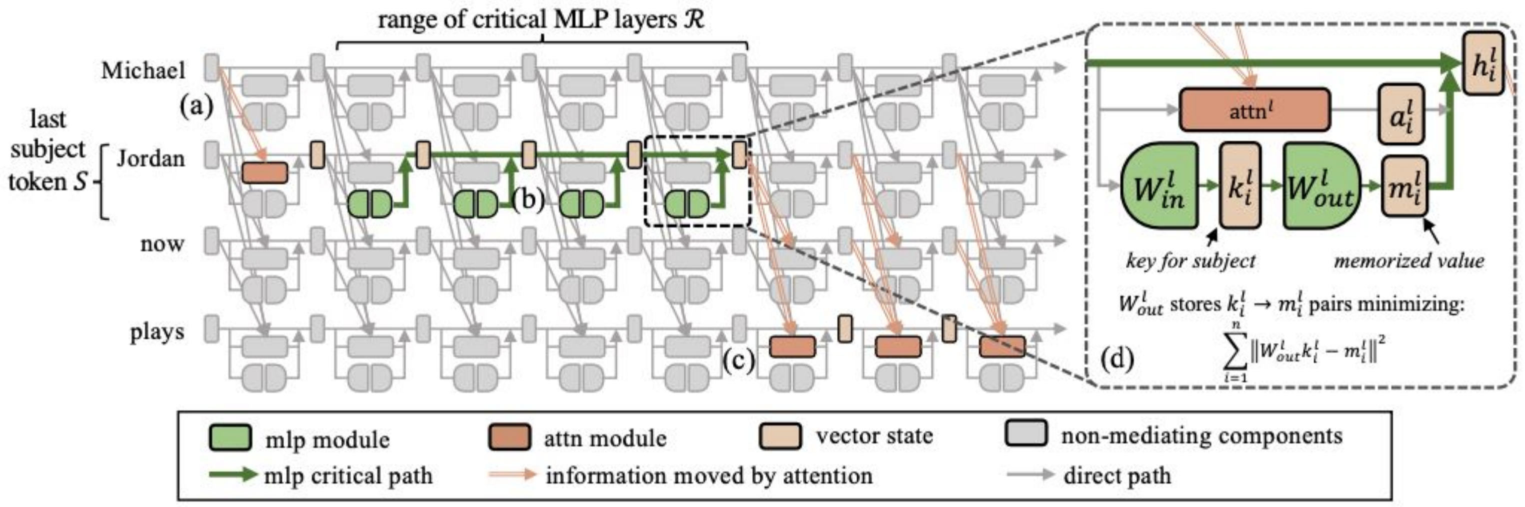


Figure 2: **MEMIT modifies transformer parameters on the critical path of MLP-mediated factual recall.** We edit stored associations based on observed patterns of causal mediation: (a) first, the early-layer attention modules gather subject names into vector representations at the last subject token S. (b) Then MLPs at layers $l \in \mathcal{R}$ read these encodings and add memories to the residual stream. (c) Those hidden states are read by attention to produce the output. (d) MEMIT edits memories by storing vector associations in the critical MLPs.

Homework - This week

1. Read the paper Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., & Bau, D. (2022, September). Mass-Editing Memory in a Transformer. In *The Eleventh International Conference on Learning Representations*. In 200 words or less write about editing memory in a transformer.

2. Run https://github.com/vijaygwu/SEAS8525/blob/main/Class_4_Transformer_Visualizations.ipynb . Explain what you see in Model view in few sentences. Feel free to comment on anything interesting that you see as well.