Natural language processing (NLP) and Attention modules.

Theories of Deep Learning: C6.5,
Lecture / Video 4
Prof. Jared Tanner
Mathematical Institute
University of Oxford

UNIVERSITY OF OXFORD
Mathematical Institute

Oxford Mathematics

Natural language processing (NLP) involves decomposing text into a finite set of tokens.

"Tokens can be thought of as pieces of words. Before the API processes the request, the input is broken down into tokens. These tokens are not cut up exactly where the words start or end - tokens can include trailing spaces and even sub-words. Here are some helpful rules of thumb for understanding tokens in terms of lengths:"

1 token $=$ 4 chars in English

1 token $=$ 3/4 words

100 tokens $=$ 75 words"

`https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them`

The Pile (Gao 20')

A public corpus for LLM pre-training

Mathematical Institute

The Pile is an open-source training corpus created by combining 22 high-quality text datasets into a single collection for language model pre-training.
Major components include

- **Web:** Filtered Common Crawl.
- **Knowledge:** Wikipedia and Stack Exchange.
- **Books:** Project Gutenberg and Books3.
- **Research:** arXiv and PubMed.
- **Code:** Public GitHub repositories.

The final corpus contains approximately 825 GB of text (about 210 billion tokens).
`https://arxiv.org/abs/2101.00027`

Tokenizer: determining the vocabulary for our language

Tokenization, vocabulary vs. length tradeoff

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units → *tokenization*
- The set of possible tokens defines the model's **vocabulary**

Tokenization ideas

- **Character-level**, e.g., "dog" → ['d', 'o', 'g']
- **UTF-8 code level**, e.g., "cafe" → [99, 97, 102, 195, 169]
- **Word-level**, e.g., "The dog runs fast." → ['The', 'dog', 'runs', 'fast', '.']
- **Subword-level**, e.g., *Byte Pair Encoding*, "happening" → ['happen', 'ing']

UNIVERSITY OF OXFORD
Mathematical Institute

Idea: Learn the vocabulary from data

Start from a fine alphabet (e.g. bytes) and iteratively *merge frequent adjacent pairs*.

BPE toy example

1. Corpus with counts:

```
"hug": 10, "pug": 5, "pun":
12, "bun": 4, "hugs": 5
```

2. Split into characters:

```
("h","u","g"): 10
("p","u","g"): 5
("p","u","n"): 12
("b","u","n"): 4
("h","u","g","s"): 5
```

First merge

3. Find the most frequent pair:

The pair

$$\texttt{("u","g")}$$

appears

$$10 + 5 + 5 = 20$$

times in the corpus.

4. Merge Rule #1:

$$\texttt{"u"} + \texttt{"g"} \rightarrow \texttt{"ug"}$$

Byte Pair Encoding (BPE): Learning the vocabulary

Mathematical Institute

After the first merge

Updated Corpus:

```
("h","ug"):  10
("p","ug"):  5
("p","u","n"):  12
("b","u","n"):  4
("h","ug","s"):  5
```

Updated Vocabulary:
Updated Vocabulary:

```
['b','g','h','n',
'p','s','u','ug']
```

The algorithm now repeats by finding the next most frequent adjacent pair in the updated corpus.

BPE Algorithm

Initialize the vocabulary V with all base bytes (or characters).

1. Count adjacent token pairs (a, b).
2. Select the most frequent pair.
3. Create a new token $c \leftarrow ab$.
4. Replace every occurrence of (a, b) with c.
5. Repeat for M merges.

Final vocabulary size:

$$|V| = |V_{\text{initial}}| + M.$$

After tokenization: IDs $\to$ vectors

- Tokenizer outputs integer IDs: $\text{tok}_t \in \{1, \ldots, V\}$.
- Show `https://tiktokenizer.vercel.app/` on "unbelievable misunderstanding"
- Convert IDs to vectors via an embedding table $E \in \mathbb{R}^{V \times d}$:

  $$\text{one-hot}(\text{tok}_t)^\top E \;=\; E_{\text{tok}_t} \in \mathbb{R}^d \quad \text{(implemented as a lookup)}.$$

- Add positional information (absolute or relative) before attention.
- Transformer maps embeddings $\to$ logits in $\mathbb{R}^V$ for next-token prediction.

Word2Vec: Learning distributed representations of words.

Classical NLP represented words using one-hot vectors, where each word in a vocabulary of size V corresponds to a basis vector in $\mathbb{R}^V$. Such vectors encode only the identity of a word, so all distinct words are equally far apart and no semantic information is represented.
Word2Vec instead learns a dense embedding

$$e(w) \in \mathbb{R}^{d_{\text{emb}}},$$

typically with $d_{\text{emb}} = 100$–1000. Words appearing in similar linguistic contexts acquire nearby vectors in the embedding space.
`https://arxiv.org/abs/1301.3781`

Given a sequence of words

$$w_1, w_2, \ldots, w_T,$$

Word2Vec learns embeddings without supervision by predicting neighbouring words.
Two related neural-network architectures were introduced.

- Continuous Bag of Words (CBOW): predict the centre word from its surrounding context.
- Skip-Gram: predict surrounding context words from the centre word.

Both models consist of only a shallow neural network, allowing training on billions of words.
`https://arxiv.org/abs/1301.3781`

Skip-Gram objective (Mikolov 13')

Learning by predicting local context

UNIVERSITY OF OXFORD
Mathematical Institute

For a context window of radius c, Skip-Gram maximises

$$\sum_{t=1}^{T} \sum_{\substack{-c \le j \le c \\ j \neq 0}} \log p(w_{t+j} \mid w_t),$$

where

$$p(w_o|w_i) = \frac{\exp(e(w_o)^T e(w_i))}{\sum_{k=1}^{V} \exp(e(w_k)^T e(w_i))}.$$

The softmax denominator requires summing over the entire vocabulary, motivating efficient approximations for large language corpora.

`https://arxiv.org/abs/1301.3781`

Negative sampling (Mikolov 13')

Learning from positive and negative examples

Consider the sentence "The cat sat on the mat." with `sat` the centre word and context radius 2. The neighbouring words define the *positive* word-context pairs

$$(\texttt{sat}, \texttt{cat}), \qquad (\texttt{sat}, \texttt{on}).$$

Rather than comparing `sat` with every word in the vocabulary, Word2Vec randomly selects a small number of unrelated words to form *negative* pairs, for example

$$(\texttt{sat}, \texttt{refrigerator}), \quad (\texttt{sat}, \texttt{democracy}), \quad (\texttt{sat}, \texttt{elephant}).$$

The network is trained to distinguish genuine neighbouring words from randomly generated ones.

`https://papers.nips.cc/paper/`

`5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf`

Negative sampling objective (Mikolov 13')

A binary classification problem

For each observed (positive) word-context pair, Word2Vec samples K random negative words from the vocabulary and minimizes $\log \phi(e_o^T e_i) + \sum_{k=1}^{K} \log \phi(-e_k^T e_i)$,
where

- e_i is the embedding of the centre word,
- e_o is the embedding of the observed neighbouring word,
- e_k is the embedding of the k^{th} randomly sampled negative word,
- $\phi(\cdot)$ is the logistic sigmoid $\phi(z) = 1/(1 + e^{-z})$.

The first term increases the similarity of true word-context pairs, while the second decreases the similarity of the $K \sim 10$ randomly generated negative pairs.

Word2Vec embeddings (Mikolov 13')

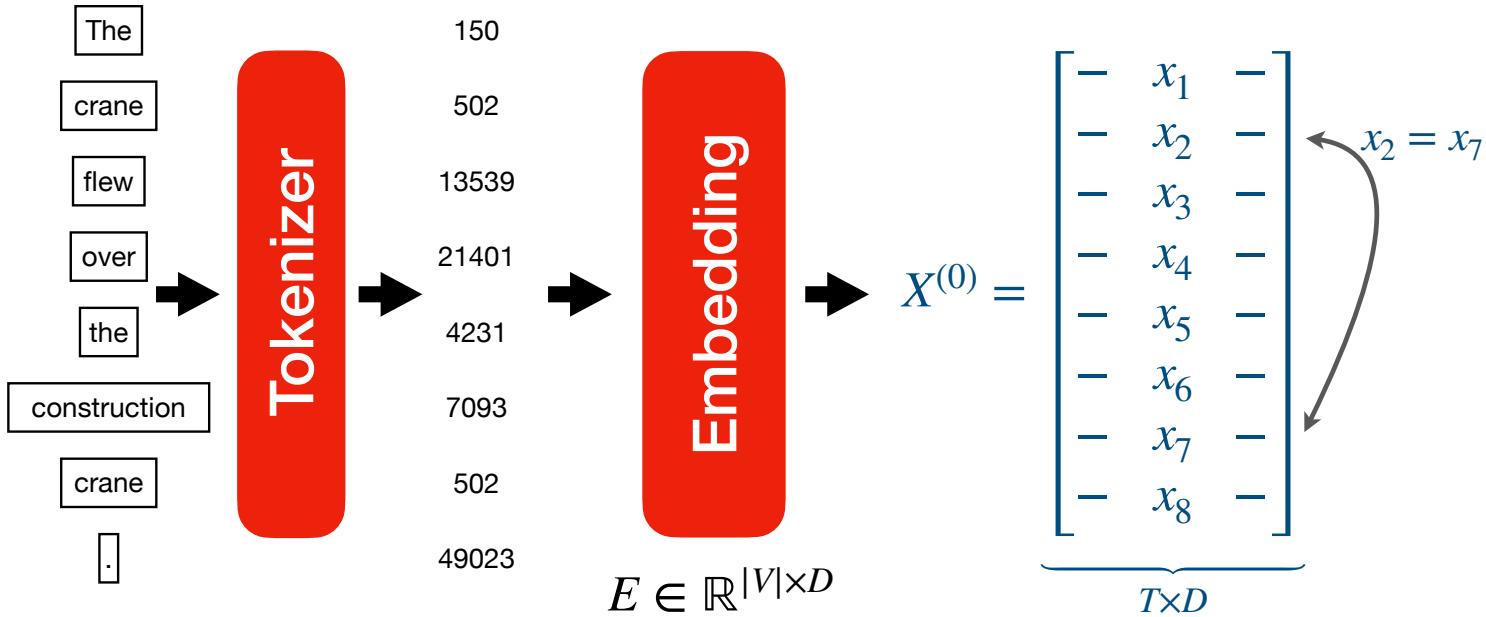
Semantic structure in vector space

After training, words with similar meanings occupy nearby locations in the embedding space.
Linear relationships between vectors often capture semantic and syntactic structure, for example

$$e(\text{King}) - e(\text{Man}) + e(\text{Woman}) \approx e(\text{Queen}).$$

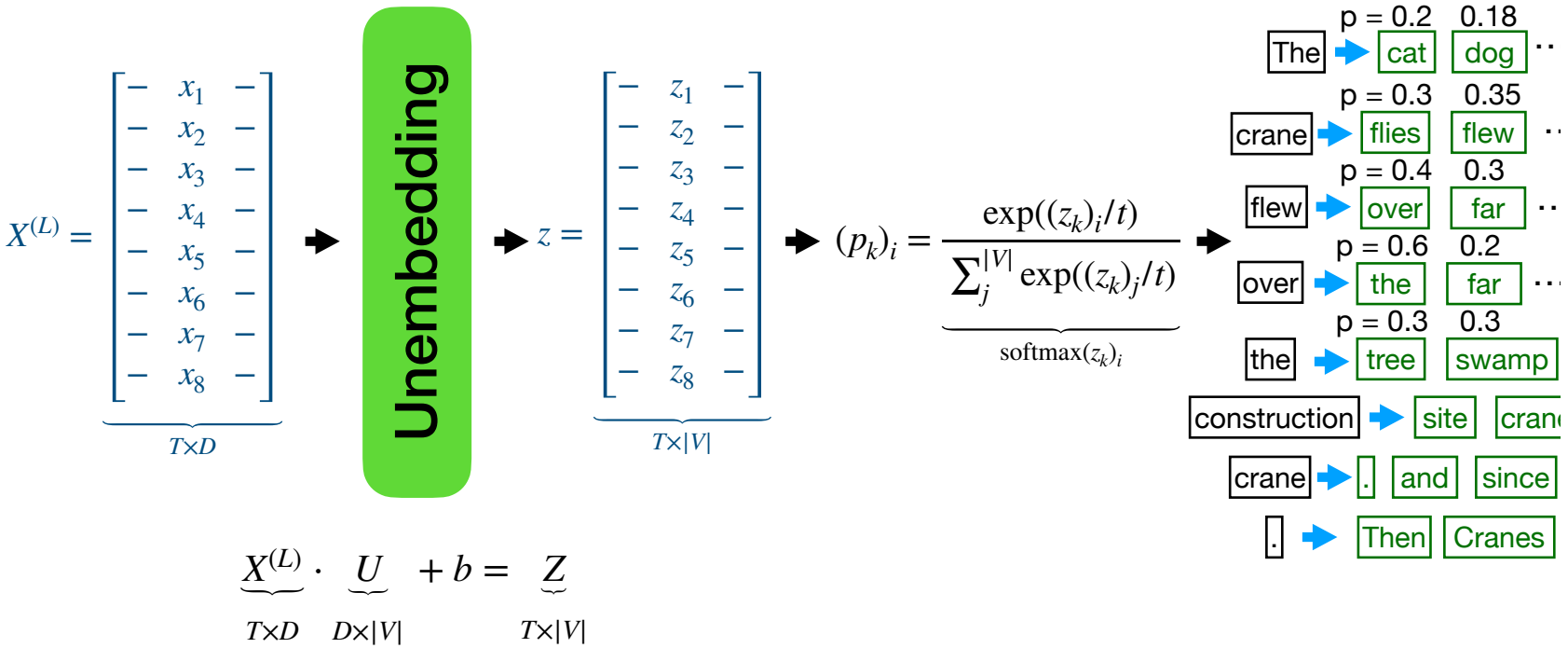
These learned embeddings became the standard input representation for many NLP models before the development of contextual transformer embeddings.
`https://arxiv.org/abs/1301.3781`

Tokenizer + Embedding

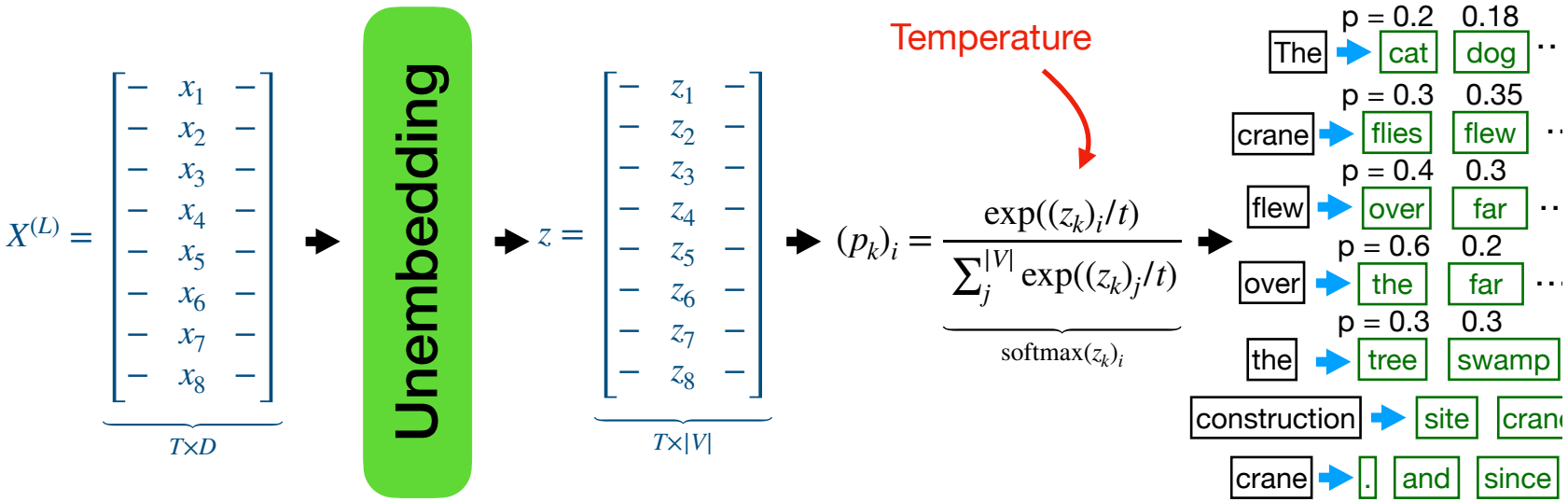


$\text{Tokenizer} : \Sigma^* \to [1,\ldots,|V|]^T \qquad \text{Embedding} : [1,\ldots,|V|]^T \to \mathbb{R}^{T \times D}$

Unembedding + softmax + training



Unembedding + softmax + training

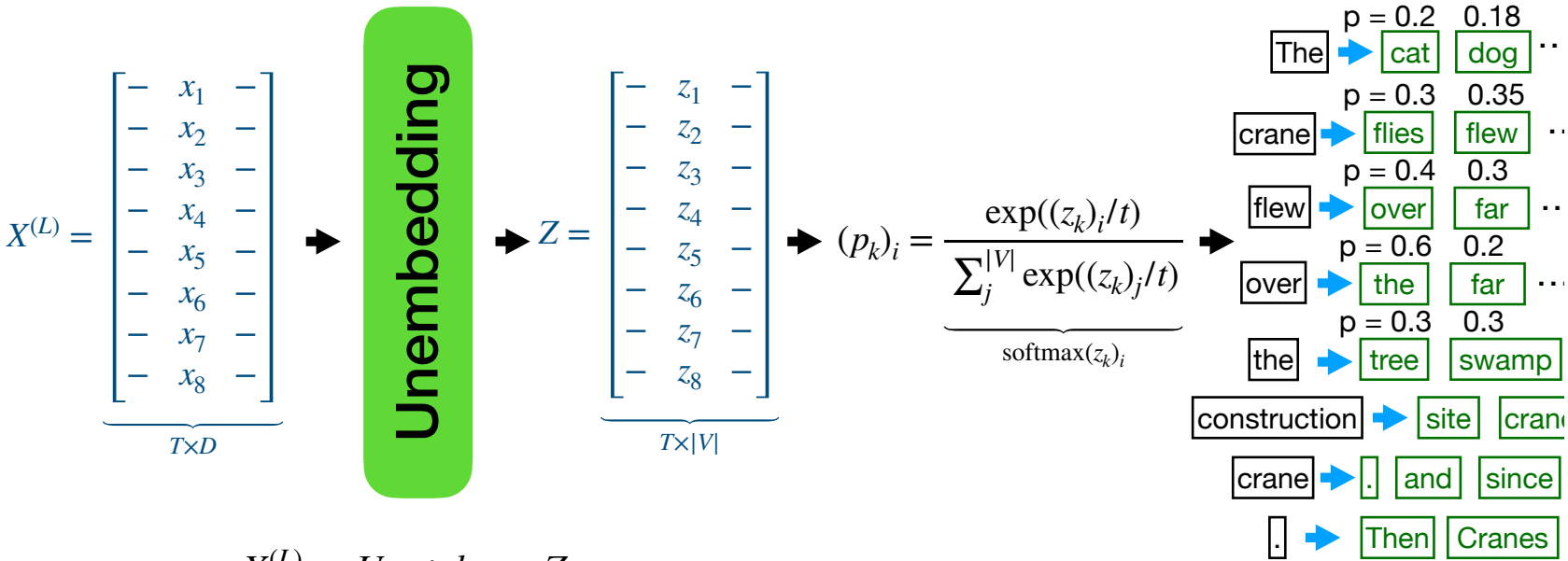


$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

Temperature:

- t < 1: logits become larger relative to each other → sharper distribution → more deterministic.
- t > 1: logits shrink → softer distribution → more random.

Unembedding + softmax + training



$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

Training loss:

$$L(\theta) = -\sum_{t=1}^{7} \log p_\theta(x_{t+1} \mid x_{1:t}) = -\sum_{t=1}^{7} \log(p_k)_{x_{k+1}}$$

Transformers and Attention, the building blocks of natural language generative models.

Attention mechanism (Vaswani 17'), equations

Key and Query quadratic form to highlight relations

UNIVERSITY OF OXFORD
Mathematical Institute

The input matrix X is then mapped to a query, key, and value matrices: $Q = XW_Q$, $K = XW_K$, and $V = XW_V$. Layer-norm is applied to the key and query matrices where each row of Q and K are normalized to have unit ℓ^2 length. These normalized query and key matrices are then compared, further normalized by scalar $d^{-\alpha}$ and typically a softmax applied row-wise

$$A = \text{softmax}\left(QK^T d^{-\alpha}\right)$$

so that each row of A has non-negative entries that sum to 1. This "self"-attention matrix is then applied to the value matrix, e.g. AV. The scaling α is typically $1/2$, this links to the softmax temperature t:

$$\text{softmax}(z; t)_i = e^{z_i/t} / \sum_k e^{z_k/t}.$$

`https://arxiv.org/abs/1706.03762`

`https://arxiv.org/abs/1607.06450`

Attention mechanism (Vaswani 17'), diagram

Key and Query quadratic form to highlight relations

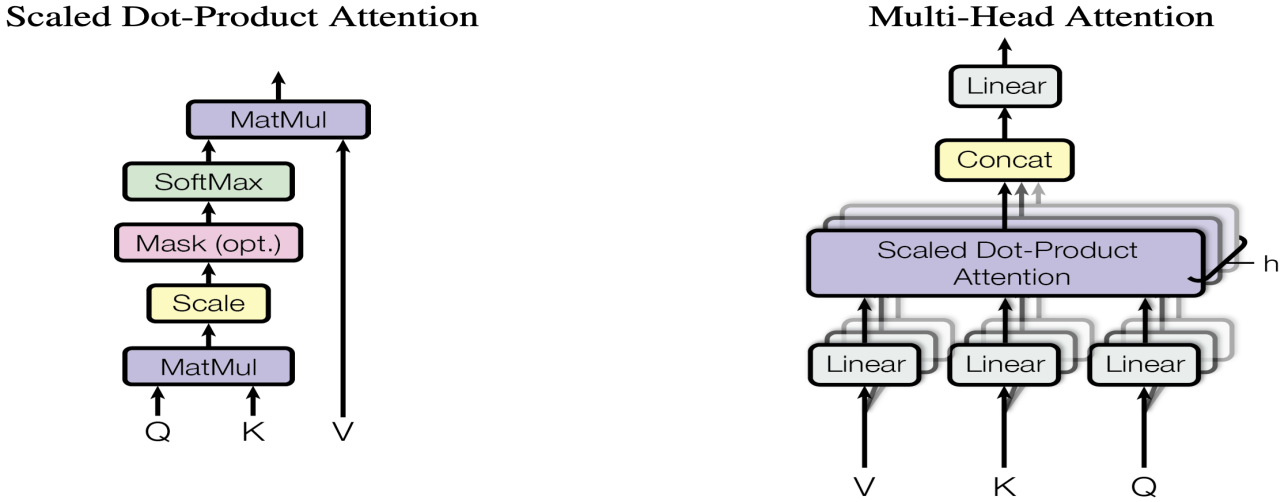


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Typically the attention mechanism applied on the left is computed within the same layer for multiple values of W_Q, W_K, and W_V with each output called a "head" and these "heads" then concatonated. `https://arxiv.org/abs/1706.03762`

Transformer mechanism (Vaswani 17'), diagram

Encoder-decoder structure using attention

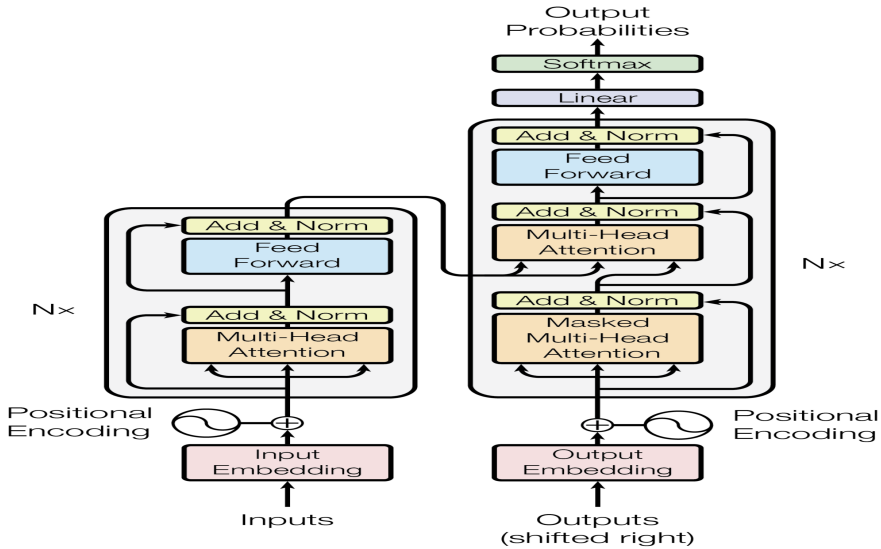


Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

The concatonated heads are then acted on by a fully connected layer W_O applied. Additional fully connected layers with ReLU are often included, as are skip connections.

`https://arxiv.org/abs/1706.03762`

GPT3 architecture (Brown 20')

GPT: Generative Pre-Trained Transformers, dimensions

Each token is then embedded in a word embedding dimension and sequentially each token in the context window is embedded as rows in a matrix $X \in \mathbb{R}^{d_{ctx} \times d_{emb}}$.
NLP networks can have tasks such as "next token prediction" where text is grown, or "translation" from one language or sentiment to another, or combined in multi-modal systems with other networks (e.g. vision) for such tasks as automatic caption generation.
By far the dominant method for NPL are attention / transformer modules.
`https://arxiv.org/abs/2005.14165`

GPT3 architecture (Brown 20'), diagram

GPT: Generative Pre-Trained Transformers

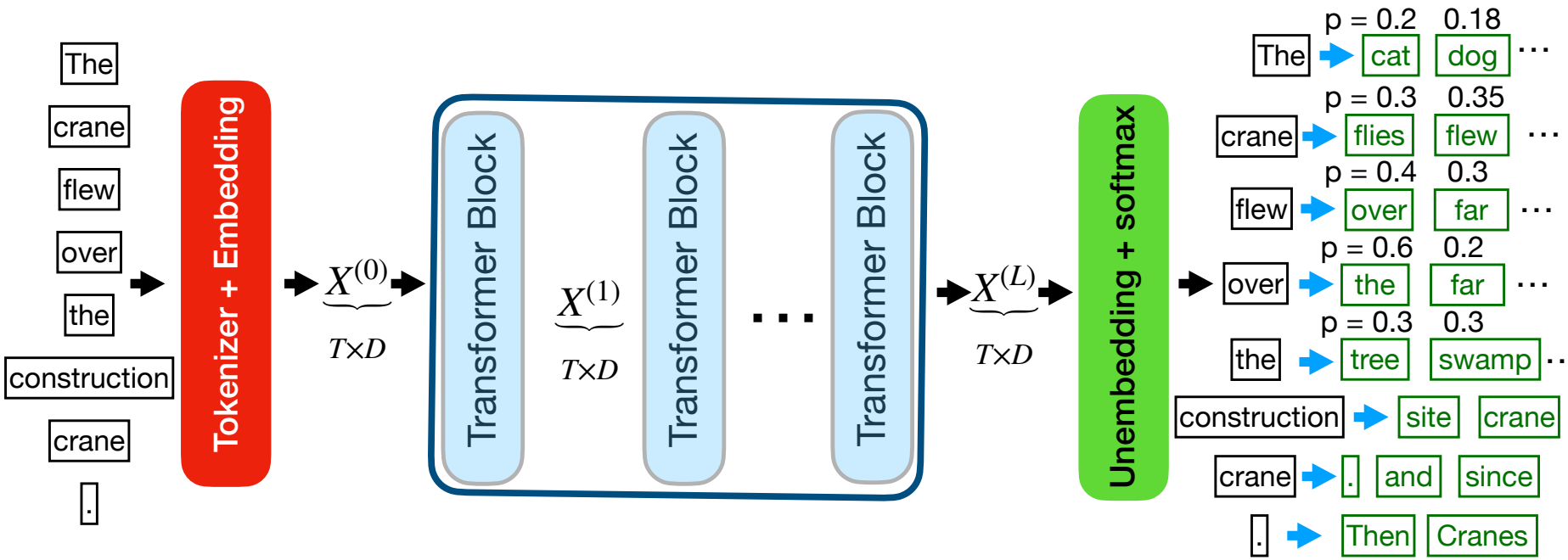
UNIVERSITY OF OXFORD
Mathematical Institute

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

Each layer of GPT3 (and earlier variants) is a transformer block. Energy usage to train and apply such large models is very costly. `https://arxiv.org/abs/2005.14165`

Summary: Decoder Transformer



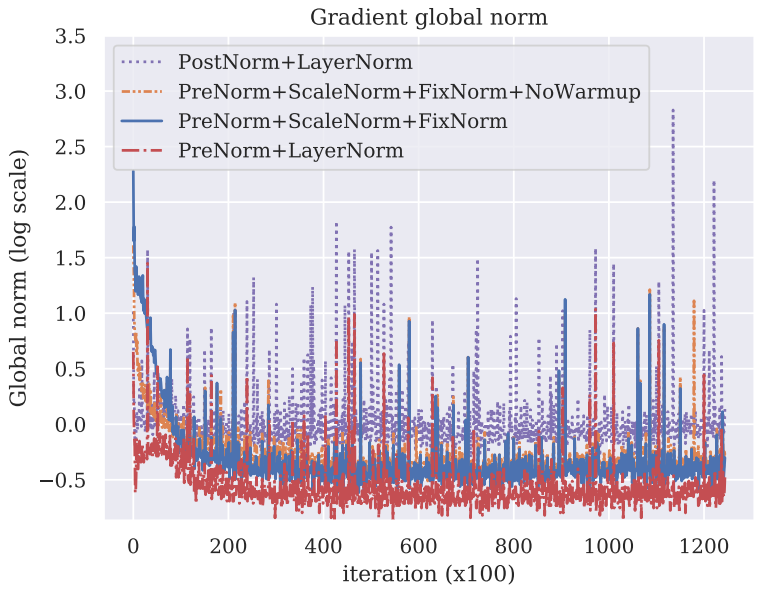
Tokenizer + Embedding : $\Sigma^* \to \mathbb{R}^{T \times D}$ string to sequence embedding

Transformer Block : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ sequence embeddings

Unembedding + Softmax : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times |V|}$ probability over token vocabulary

Normalization / Stability

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.



Main role in Transformers Keep each sublayer input well-conditioned before attention / FFN (MLP) computations.

Pre-norm vs post-norm

Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

Pre-norm (modern LLMs)

$$x_{l+1} = x_l + F(\mathrm{Norm}(x_l))$$

- *Before* MHA (or FFN).
- Cleaner residual path for gradients.
- More stable for deep, large-scale training.

Most modern LLMs use **pre-norm** because it trains more reliably at scale.

LayerNorm vs RMSNorm

LayerNorm normalizes both mean and variance across features:

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \varepsilon}} + \beta$$

RMSNorm normalizes only the root-mean-square:

$$\text{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}}$$

Typical examples (LayerNorm):
Original Transformer, BERT, GPT-2/3, OPT

Typical examples (RMSNorm):
T5, PaLM, LLaMA-family, Gemma

Pre-vs-post Layer normalization (Xiong et al. 20')

Bulk normalization hyperparameters

Mathematical Institute

Layer-norm acts as batch-norm, but as the input is a matrix $X \in \mathbb{R}^{d_{ctx} \times d_{emb}}$ it can act on one input, but for each row in X. It can be applied at X fully (pre-) or just before the attention a FFN layer (post-) layer norm:

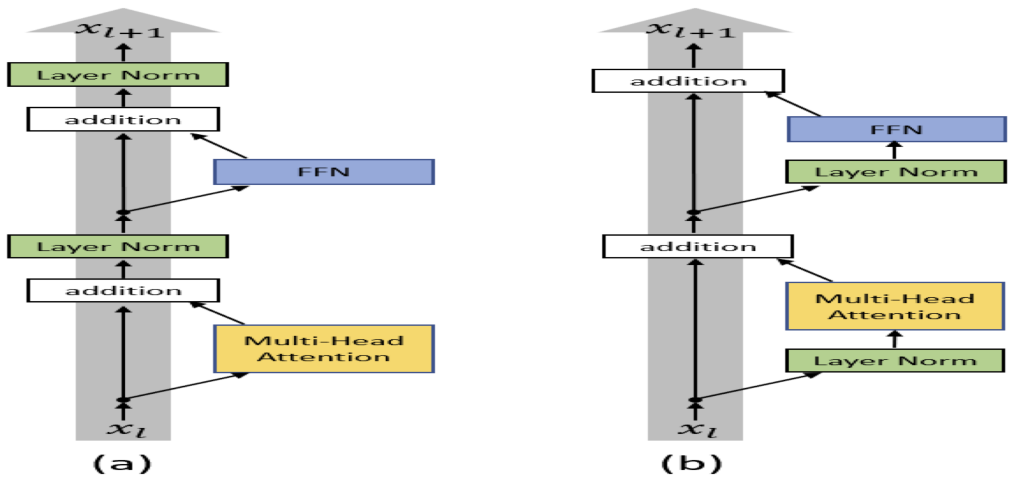


Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

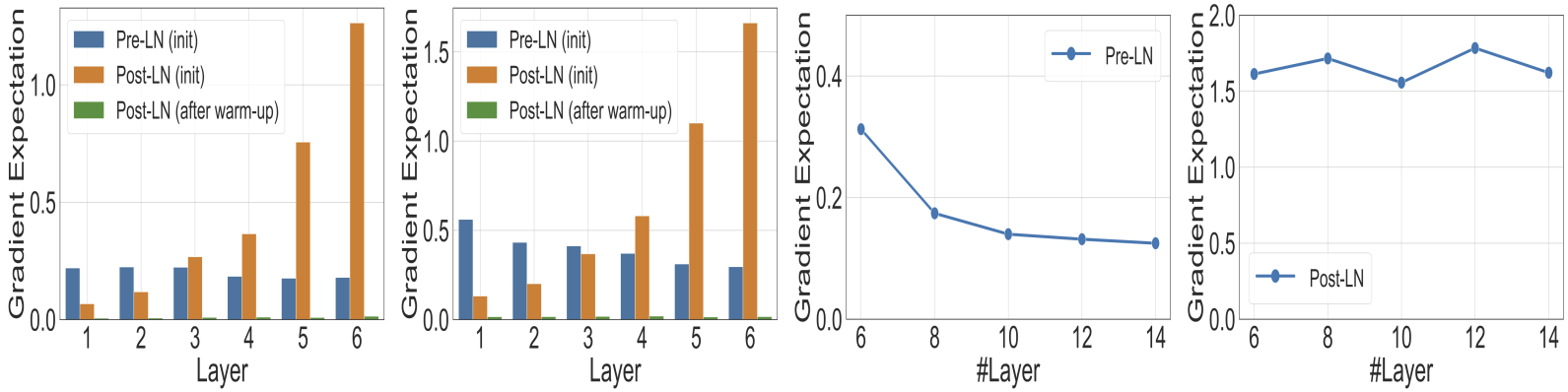
`https://arxiv.org/pdf/2002.04745`

Pre-vs-post Layer normalization (Xiong et al. 20')

The location of the layer-norm and gradient values

UNIVERSITY OF OXFORD
Mathematical Institute

The location of layer-norm has a substantial impact on the size of gradients per layer. This can be further impacted in "warm-up" training is done with stepsize initially starting small, then growing linearly to a maximum value.



(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers (c) Pre-LN Transformer (d) Post-LN Transformer

Figure 3. The norm of gradients of 1. different layers in the 6-6 Transformer (a,b). 2. $W^{2,L}$ in different size of the Transformer (c,d).

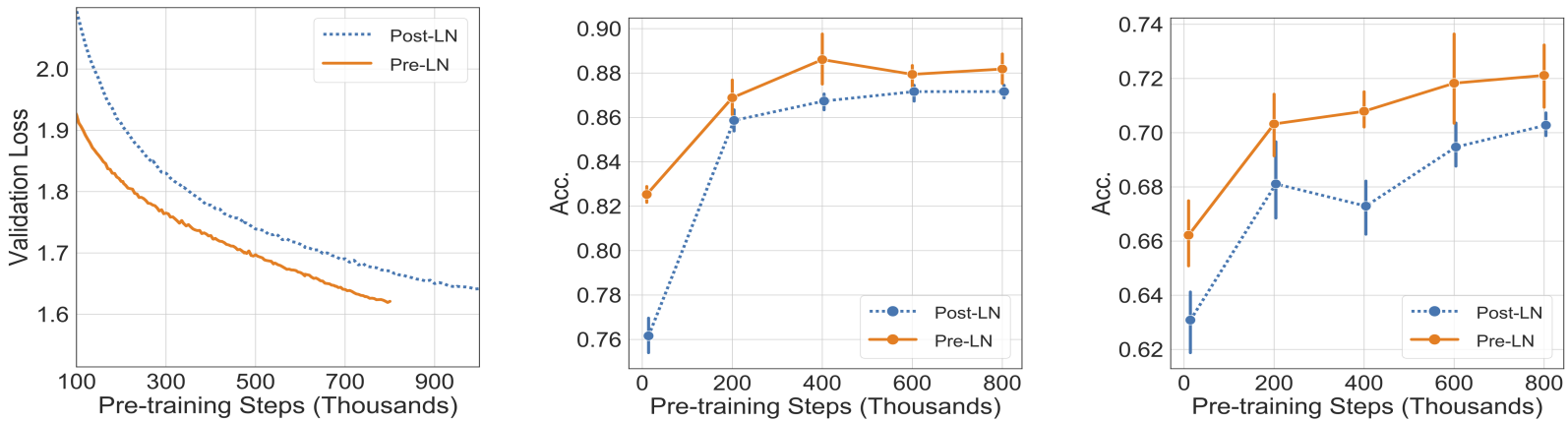
`https://arxiv.org/pdf/2002.04745`

Pre-vs-post Layer normalization (Xiong et al. 20')

The resulting impact on loss and accuracy

Mathematical Institute

Pre-layer-normalization (acts on all of X) shows reduced loss and superior accuracy for translation (BERT), paraphrasing (MRPC), and if sentences are "entangled" (RTE).



(a) Validation Loss on BERT (b) Accuracy on MRPC (c) Accuracy on RTE

Figure 5. Performances of the models on unsupervised pre-training (BERT) and downstream tasks

`https://arxiv.org/pdf/2002.04745`

Takeaways: Modern Transformer Decoder Block

UNIVERSITY OF OXFORD
Mathematical Institute

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU $\Rightarrow$ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit + $\text{Swish}(x)$.
- **Parameters:** Biases $\Rightarrow$ **Remove Biases**
  - Bias terms $(+b)$ are dropped to save memory and increase throughput.

Transformers and Attention are also now used for Vision tasks.

Vision Transformer - ViT (Dosovitskiy 21'), diagram

Patches of images embedded and acting as token; with position embedding

University of Oxford
Mathematical Institute

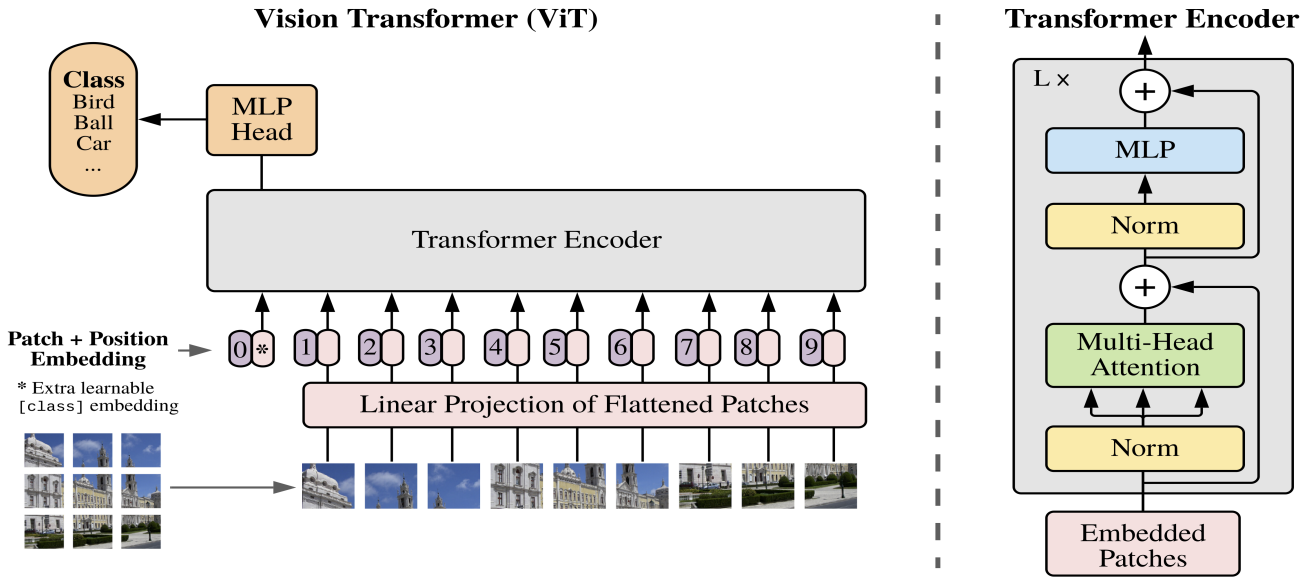


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

https://arxiv.org/pdf/2010.11929

Vision Transformer - ViT (Dosovitskiy 21'), formulae

Positional embedding included

The MLP contains two layers with a GELU non-linearity.

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\, \cdots;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

Inductive bias. We note that Vision Transformer has much less image-specific inductive bias than CNNs. In CNNs, locality, two-dimensional neighborhood structure, and translation equivariance are baked into each layer throughout the whole model. In ViT, only MLP layers are local and translationally equivariant, while the self-attention layers are global. The two-dimensional neighborhood structure is used very sparingly: in the beginning of the model by cutting the image into patches and at fine-tuning time for adjusting the position embeddings for images of different resolution (as described below). Other than that, the position embeddings at initialization time carry no information about the 2D positions of the patches and all spatial relations between the patches have to be learned from scratch.

`https://arxiv.org/pdf/2010.11929`