

Introduction to Visual Transformers and Transformer Attributions

Filip Kołodziejczyk



MI2.AI Seminar, Warsaw, May 13th, 2024

Agenda

- Recap of Transformer architecture
- Introduction to Visual Transformers
- Explaining the Transformers with Attention
- Review of *Deep Learning: Foundations and Concepts*

Recap of Transformer Architecture

based on

*Deep Learning: Foundations
and Concepts*

by C.M. Bishop and H. Bishop

Simple definition

Transformer transforms a set of vectors in some representation space into a corresponding set of vectors in some new space. We wish this new space to have a richer internal representation. For that, we use chained transformer layers, which comprise two stages: the attention mechanism, mixing corresponding features, and 2nd stage, processing each row independently.

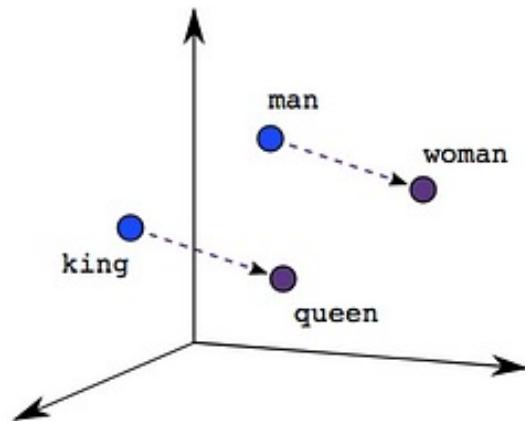
$$\tilde{X} = \text{TransformerLayer}[X]$$

Promises it gives:

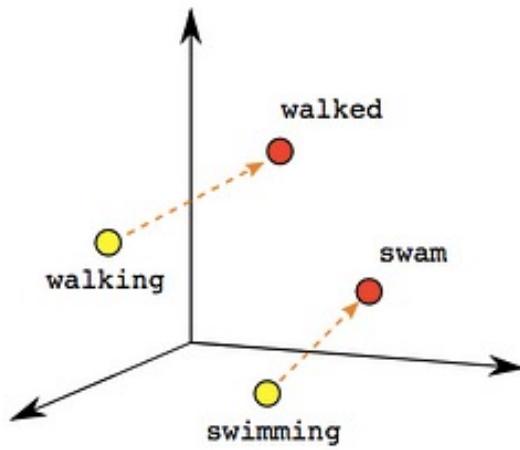
- Very effective transfer learning, foundational models reused for many downstream tasks via fine-tuning.
- Possible to train via self-supervised learning to process vast amounts of data.
- Enables parallel processing, so huge models are possible to create.

Transformers, initially created for NLP tasks, achieved SOTA in many domains.

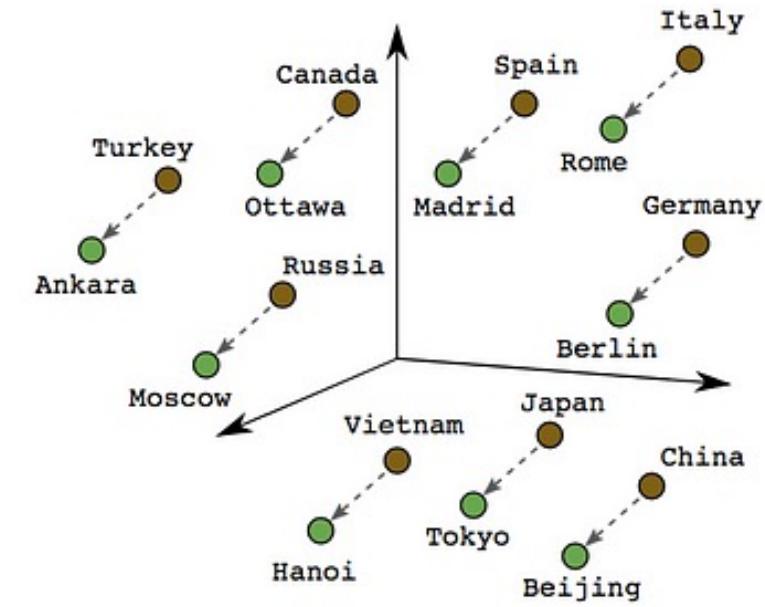
Before the transformers (word embeddings)



Male-Female



Verb Tense



Country-Capital

Source: [A Guide to Word Embedding. What are they? How are they more useful... | by Shraddha Anala | Towards Data Science](#)

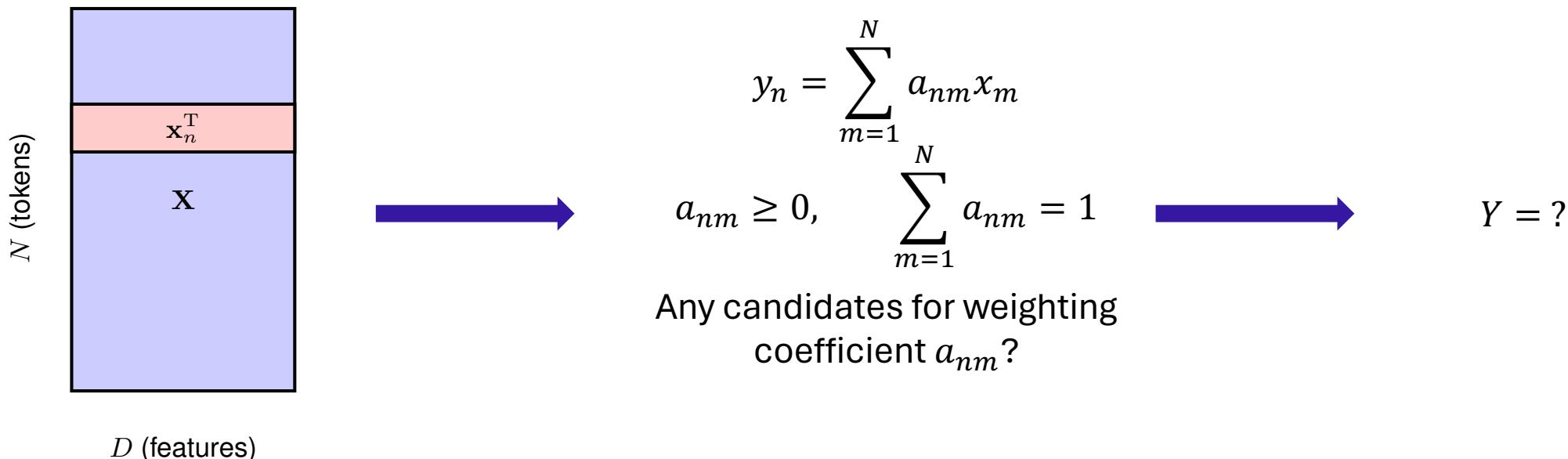
Problem with word embeddings

Word embedding always maps a given word to the same embedding vector, but words **change meaning influenced by context**. NN should attend to, i.e., rely more on some words from the rest when determining the interpretation of the word of our interest. Moreover, **those words' position may vary from sentence to sentence**.

*I swam across the river to get to the other bank.
I walked across the road to get cash from bank.*

Attention as solution (naïve approach)

Transformation layer should transform input – set of vectors (called tokens) $\{x_1, \dots, x_n\} \in \mathbb{M}^{N \times D}$ (N – #tokens, D – #features) into richer form of embedding, $\{y_1, \dots, y_n\}$ where each y_i depends on all input vectors $\{x_1, \dots, x_n\}$.



Information retrieval will help

Problem: Choose which movie to watch on an online movie streaming service.

Solution: Associate each movie with a list of attributes. A user can then provide a list of desired characteristics. We can find the film with the best attributes that match the desired ones. We can even measure the degree of match for each one. As a result, we return this movie. We call the list of movie attributes *keys K*, user list a *query Q*, and, movies corresponding to keys – values *V*.

Mapping to the attention problem:

How to measure the similarity between the x_n and x_m ?

$V - x_n$

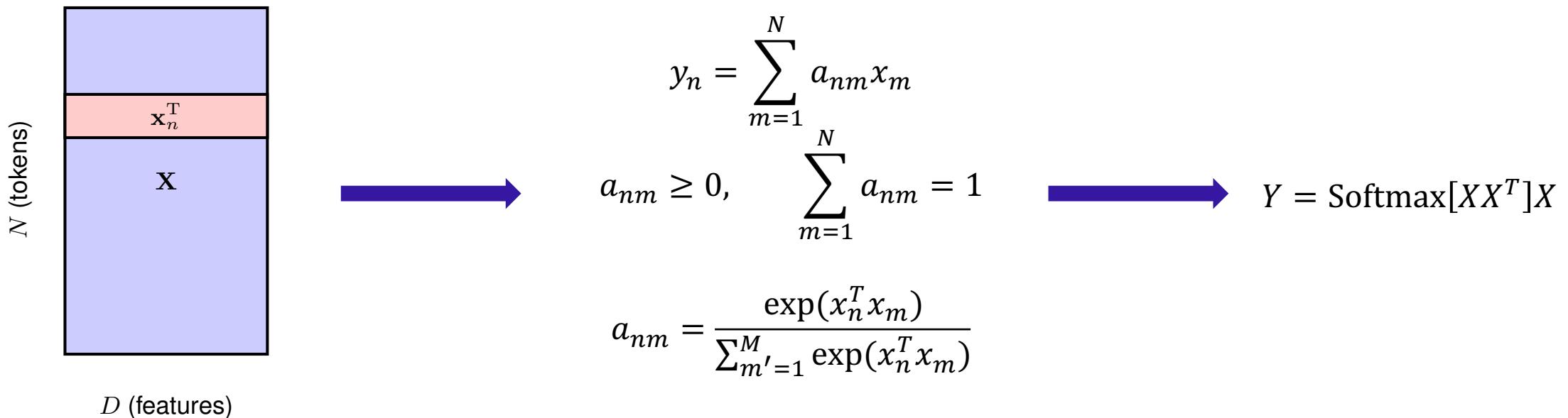
$K - x_n$

$Q - x_m$

We use the dot product between keys and queries, i.e. $x_n^T x_m$.

Attention as solution (naïve approach)

The transformation layer should transform input – set of vectors (called tokens) $\{x_1, \dots, x_n\} \in \mathbb{M}^{N \times D}$ (N – #tokens, D – #features) into the richer form of embedding, $\{y_1, \dots, y_n\}$ where each y_i depends on all input vectors $\{x_1, \dots, x_n\}$.



Attention as solution - challenges

Issue 1: The transformation is fixed; nothing can be learned. Also, each of the feature values is equally important.

Solution: Introduce a weight parameters $U \in \mathbb{M}^{D \times D}$. Then, we can use $\tilde{X} = XU$.

$$Y = \text{Softmax}[XX^T]X \quad \longrightarrow \quad Y = \text{Softmax}[XUU^TX^T]XU$$

Issue 2: Matrix XUU^TX^T is symmetric, while the attention mechanism should support asymmetry. E.g. The *chisel* should be strongly related to the *tool* but not vice versa (as there are many tools).

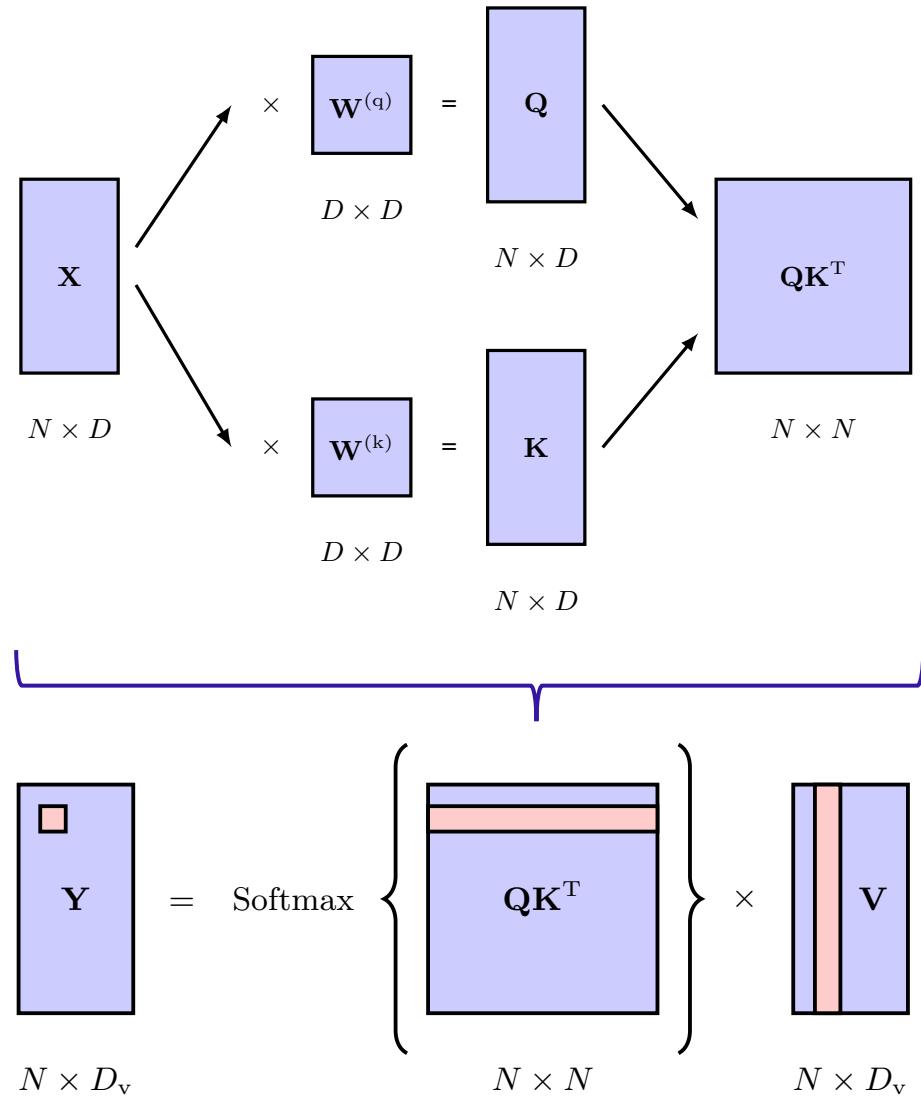
Solution: Introduce separate weights $W^{(q)}$, $W^{(k)}$, $W^{(v)}$ and use $Q = XW^{(q)}$, $K = XW^{(k)}$, $V = XW^{(v)}$.

$$Y = \text{Softmax}[XUU^TX^T]XU \quad \longrightarrow \quad Y = \text{Softmax}[QK^T]V$$

Self-attention (result)

The name comes from using the same sequence for Q , K , and V .

We often add bias by augmenting X with a column of 1's.



Self-attention - challenges

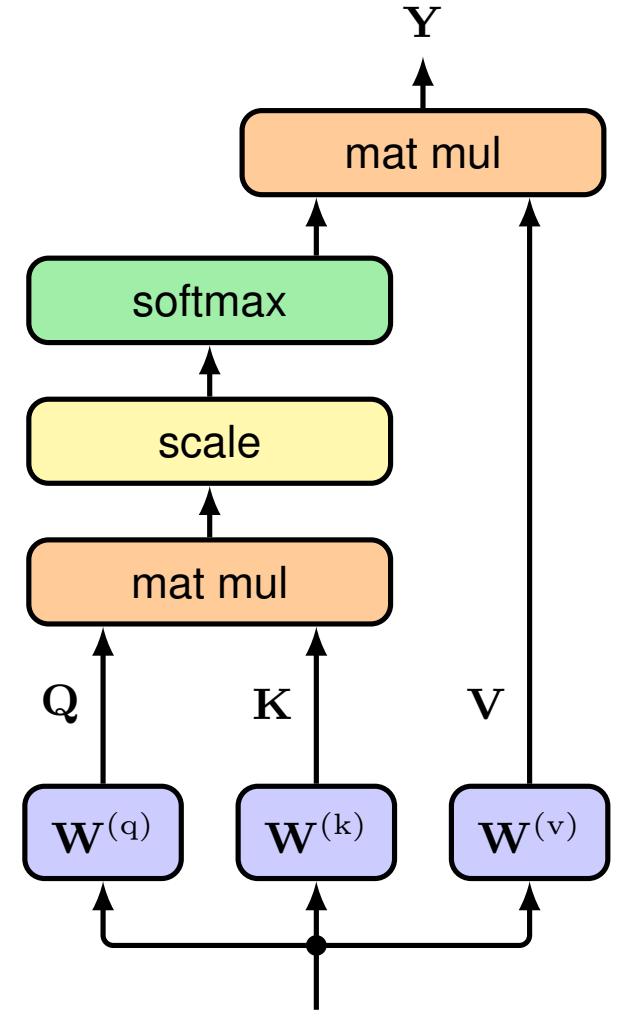
Issue 3: The gradient of softmax becomes exponentially small for high-magnitude inputs.

Solution: Rescale the QK before the softmax function. Assuming Q and K are the i.i.d. vectors with 0 mean and variance, then their dot product variance would be D .

$$Y = \text{Softmax}[QK^T]V$$



$$Y = \text{Attention}(Q, K, V) \stackrel{\text{def}}{=} \text{Softmax} \left[\frac{QK^T}{\sqrt{D}} \right] V$$



Final form (attention head)

Multi-head attention

Problem: Multiple attention patterns might be relevant simultaneously (e.g., in NLP, some patterns apply to tense, others associated with vocabulary). A single head might average those patterns.

Solution: Use multiple attention heads in parallel, allowing each to focus on its pattern. We define heads as

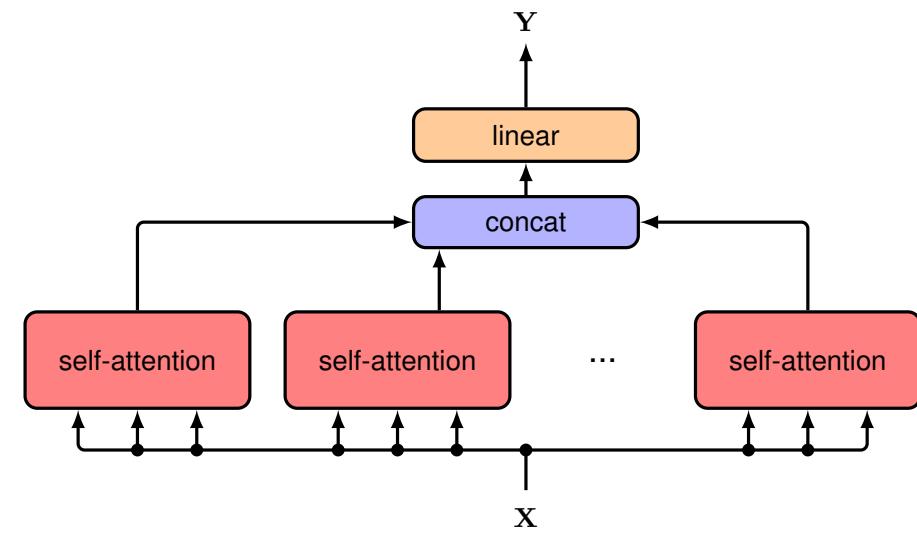
$$H_h = \text{Attention}(Q_h, K_h, V_h)$$

Where each weight matrix $W_h^{(q)}$, $W_h^{(k)}$, $W_h^{(v)}$ is independent. To combine those heads to get single output, we concatenate them and linearly transform using matrix $W^{(\circ)}$ (learned along with weight matrices) to the desired dimension.

$$Y(X) = \text{Concat}[H_1, \dots, H_H]W^{(\circ)}$$

$$\begin{matrix} \mathbf{H}_1 & \mathbf{H}_2 & \dots & \mathbf{H}_H \end{matrix} \times \mathbf{W}^{(\circ)} = \begin{matrix} \mathbf{Y} \end{matrix}$$

$N \times HD_v$ $HD_v \times D$ $N \times D$



Transformer layer

- *Residual connections* are utilized to improve the training (stability and performance). Other training efficiency improvements come from *layer normalization*.

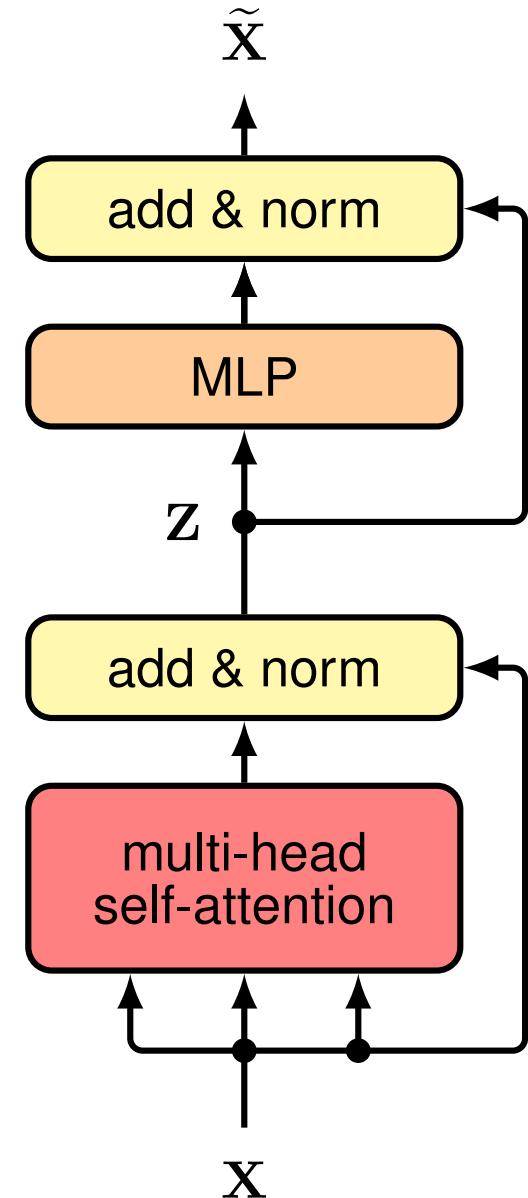
$$Z = \text{LayerNorm}[Y(X) + X]$$

- Sometimes, the order of operations is reversed. This can result in more effective optimization).

$$Z = Y(\text{LayerNorm}[X]) + X$$

- Finally, we can increase the layer flexibility by post-processing the attention result with a standard, fully connected network (multi-layer perceptron), which processes each vector one by one. Again, *layer normalization* and *residual connections* are used (in reversed order, too).

$$\tilde{X} = \text{TransformerLayer}[X] = \text{LayerNorm}[\text{MLP}[Z] + Z]$$



Positional encoding

Issue: Transformer is equivariant to input permutations. Lack of dependence on token order limits the sequential capabilities.

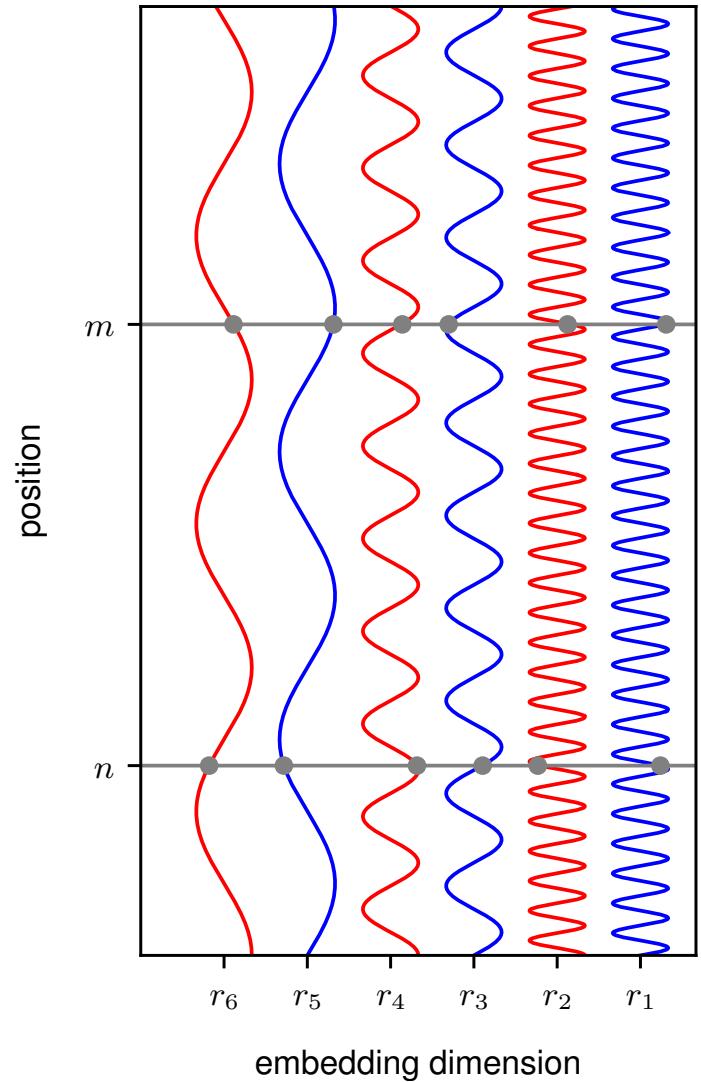
Solution: Construct the position encoding vector r_n and add it to the token vector:

$$\tilde{x}_n = x_n + r_n$$

Thanks to the residual connections, position encoding does not get lost through the consecutive layers.

Positional encoding should be invariant to the sequence length and unique for each position. A common solution is a technique based on sinusoidal functions (i – position, L – input sequence length):

$$r_{ni} = \begin{cases} \sin\left(\frac{n}{L^{i/D}}\right), & \text{if } i \text{ is even} \\ \cos\left(\frac{n}{L^{(i-1)/D}}\right), & \text{if } i \text{ is odd} \end{cases}$$



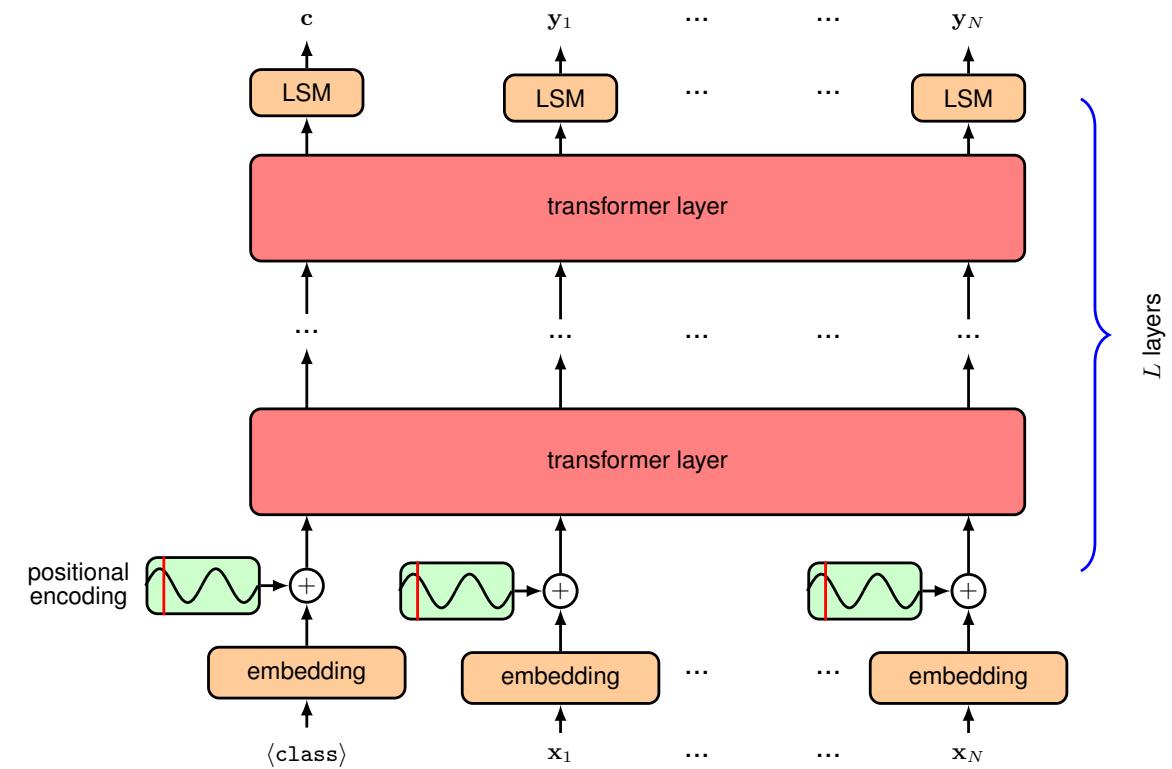
Transformer architecture categories

Depending on the form of the input and output of the model, we distinguish 3 types of transformers:

- **Encoder** – sequence as input and single variable as output
- **Decoder** – single vector as input and sequence as output
- **Sequence-to-sequence** – both the input and output comprise a sequence of variables

Encoder transformers

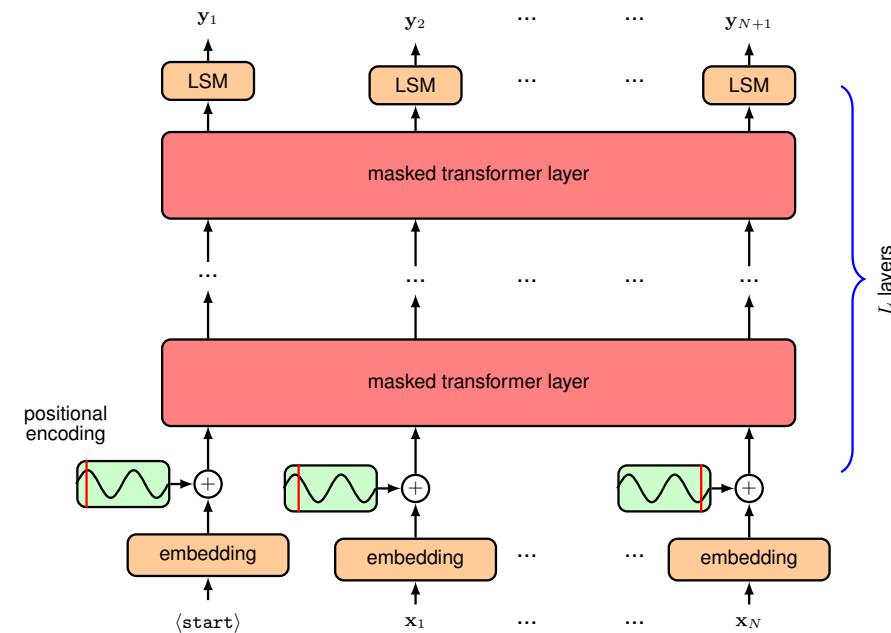
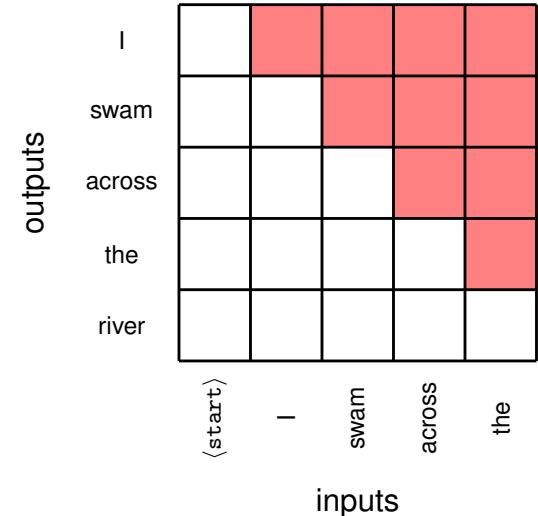
- Uses standard transformer layers, as presented in the previous slides – self-attention is bidirectional, i.e., uses both preceding and following tokens concerning the current token
- Sample use cases: classification, missing word prediction, Named Entity Recognition
- Sample foundational model: [BERT](#)



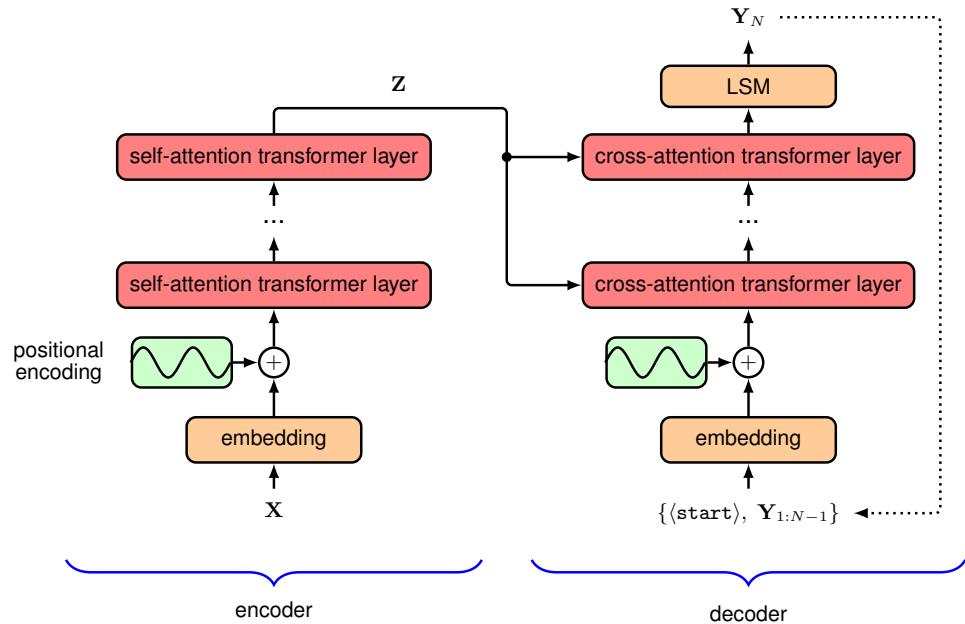
Decoder transformers

- Main goal is predict future values based on past values (autoregressive model)
- Uses masked transformer layers – self-attention is unidirectional, i.e., uses only preceding tokens. Thanks to that, the model can still be trained in parallel using the whole sequence without cheating: knowledge about the following tokens.
- Masking is done by setting the attention weight of all the following tokens to 0 or $-\infty$.
- While training is parallelized, output generation is sequential
- Sample use cases: next word prediction, sentence completion, text-to-speech
- Sample foundational model: [GPT-3](#)

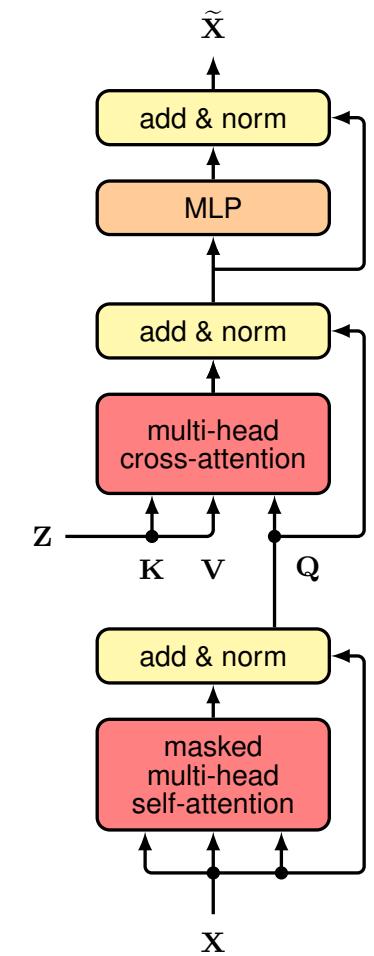
Masking



Sequence-to-sequence transformers



- Uses both an encoder and a decoder
- The encoder consumes the input sequence and creates its representation
- Then, the decoder generates an output sequentially, using the so-far generated output
- The decoder's transformer layer is modified: after the masked self-attention layer, there is an additional cross-attention layer
- Cross-attention layer combines the representation of the input sequence created by the encoder with the decoder's masked-layer output
- Sample use cases: machine translation, Q&A, image captioning
- Sample foundational model: [BART](#)



Cross-Attention Layer

Introduction to Visual Transformers

based on

*Deep Learning: Foundations
and Concepts*

by C.M. Bishop and H. Bishop

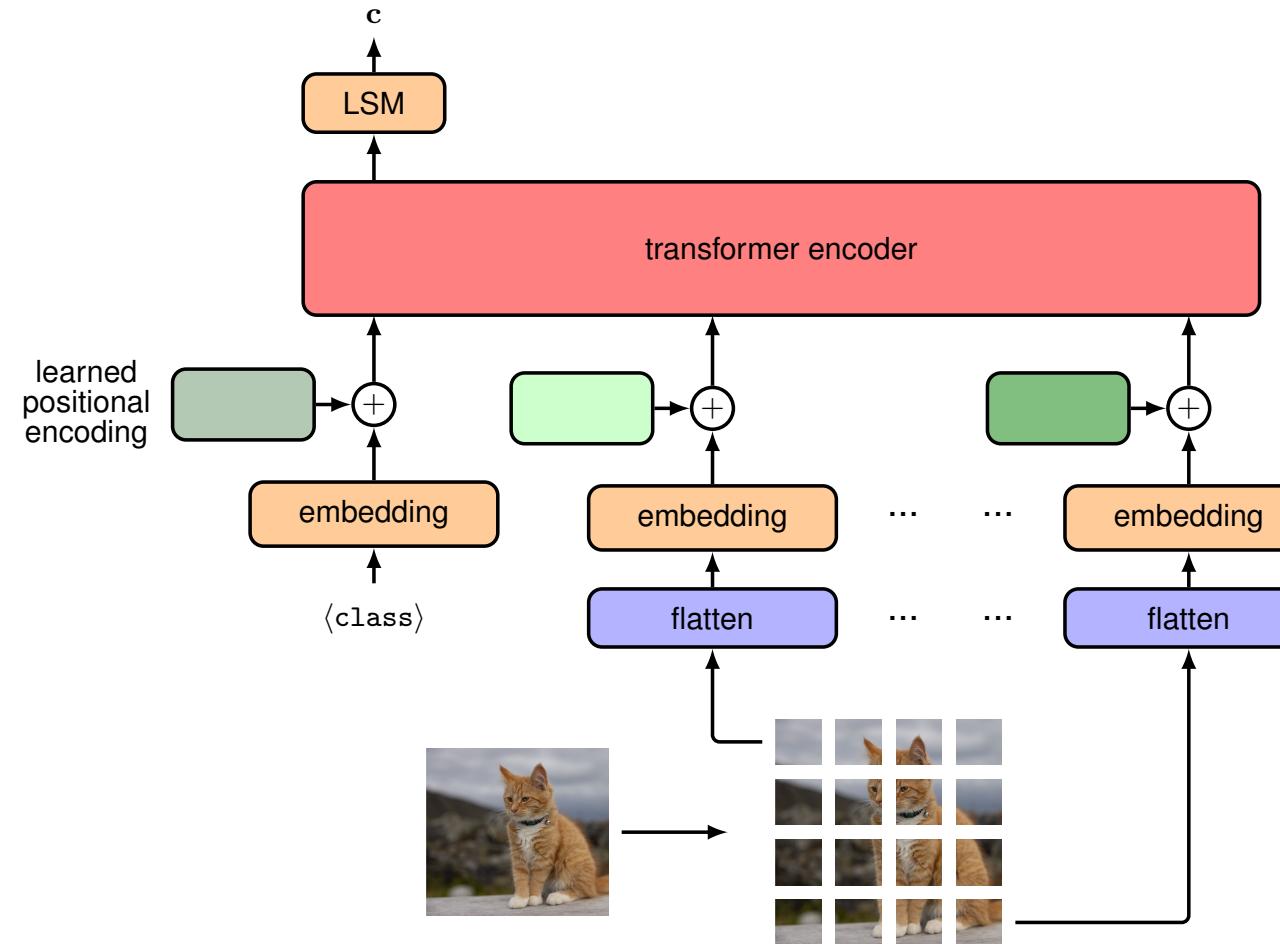
Vision transformers

Transformers make very few assumptions about input data. Therefore, they can be used for a variety of tasks. One of them is CV, where it achieved a SOTA performance. The most common choice is the encoder transformer. This approach is called Vision Transformer (ViT).

When it comes to processing images with transformers, a crucial step is tokenization. The most straightforward approach is to use each pixel as a token. However, the memory requirements of a standard transformer increase quadratically with the number of input tokens, posing a significant challenge. This complexity underscores the need for innovative solutions. Two most common are:

- Split images $x \in \mathbb{R}^{H \times W \times C}$ into a set of patches of the same size. Each image is split into non-overlapping patches of size $P \times P$ and then flattened into a one-dimensional vector, which gives representation $x_p \in \mathbb{R}^{N \times (P^2 C)}$ where the total number of patches for images is $N = \frac{HW}{P^2}$.
- Feed the image into a small CNN. This will downsample the image to the manageable number of tokens.

Vision transformers



Explaining the Transformers with Attention

based on

*Quantifying Attention Flow in
Transformers*

by S. Abnar and W. Zuidema

Authors

- **Samira Abnar** – Apple ML research team member and PhD candidate at the University of Amsterdam; research interests: ML and Cognitive Science
- **Willem Zuidema** - associate professor of NLP, explainable AI , and cognitive modelling at the University of Amsterdam; research interests: NLP, Interpretability, XAI, Cognitive Modelling, Evolution of Language

Setups and Problem Statement

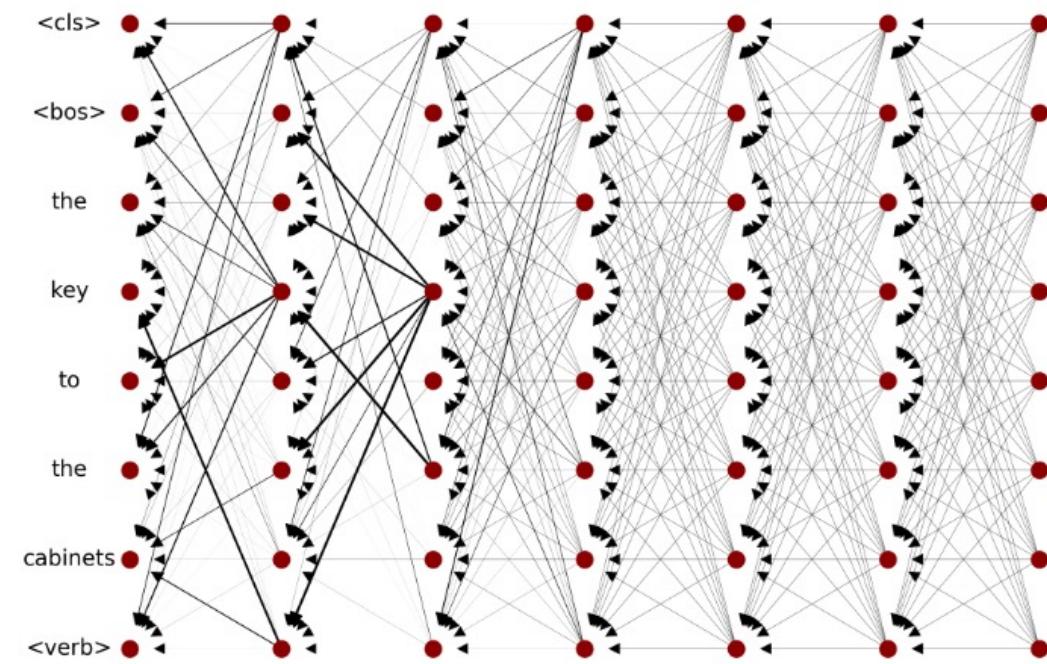
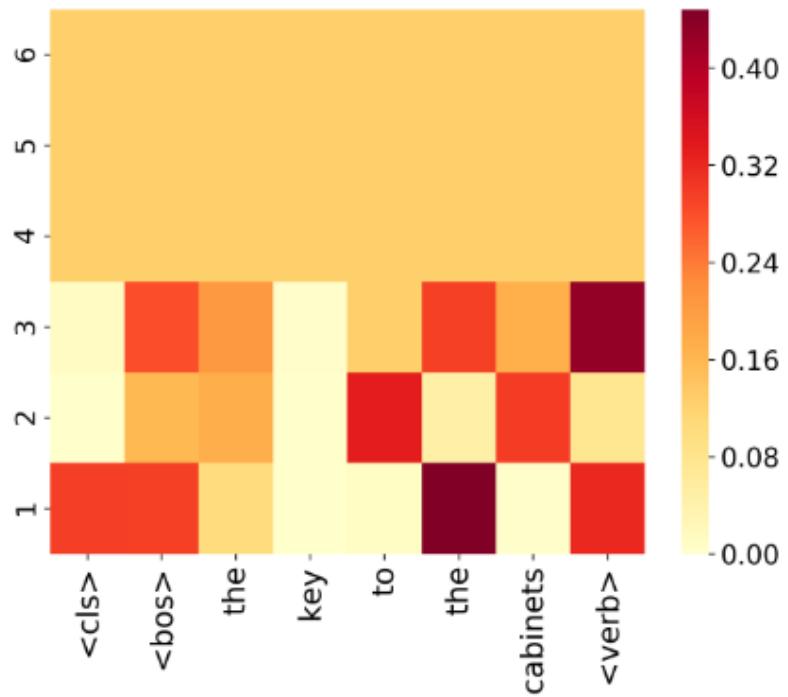
- **Problem:** Verb number prediction task - predicting singularity or plurality of a verb of a sentence, where the input is the sentence up to the verb position
- **Model:** transformer encoder, with GPT2 Transformer blocks (w/o masking), 6 layers, 8 heads, embedding size of 128, using CLS [classification] token
- **Achieved accuracy:** 0.96

Self-Attention vs Explainability

- Attention visualizations as easy and very popular way to interpret the model
- Due to the nature of self-attention, across the layers of model different tokens get increasingly mixed
- **Which part of the input is the most important?**

Raw Attention

Visualization of correctly classified example. In higher layers, there is a lack of interpretable pattern.

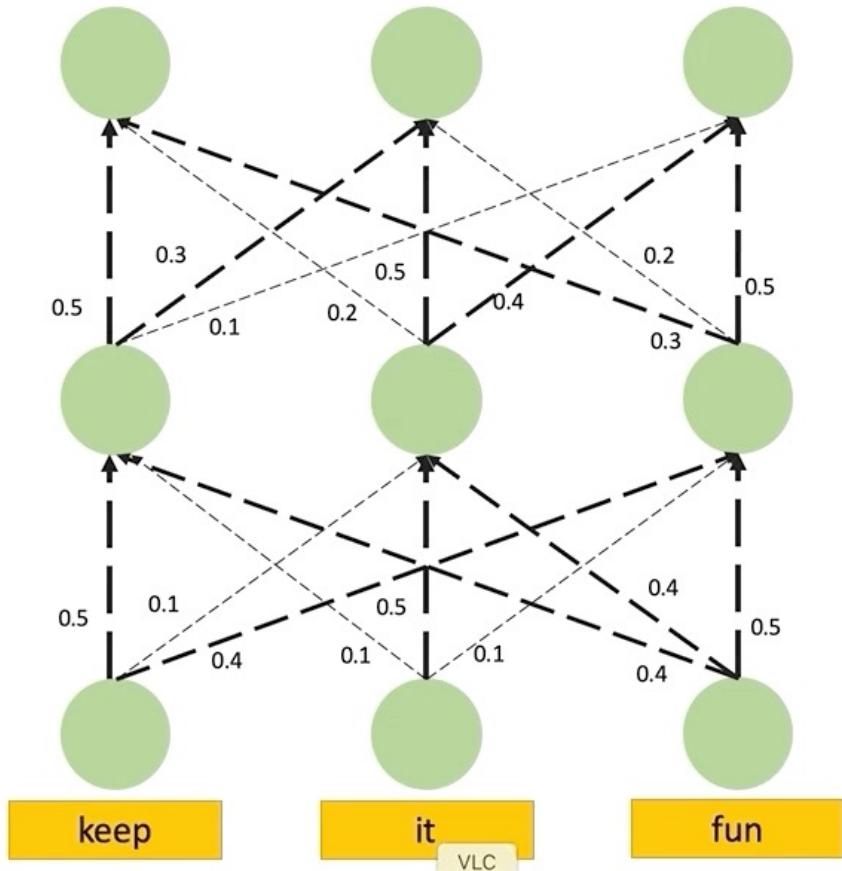


General concept

- Model information flow with a DAG:
 - nodes = input tokens/hidden embeddings
 - edges = attentions for the nodes to the previous layer
- Recursively compute token attentions in each layer
- Include residual connections
- Multi-head attention via averaging the attention (*)
- Computing values in layer $l + 1$:
$$V_{l+1} = V_l + W_{att}V_l, \quad (W_{att} \text{ —attention matrix})$$
- Only for visualization and debugging, no impact on the model itself (applied post hoc)

Attention rollout

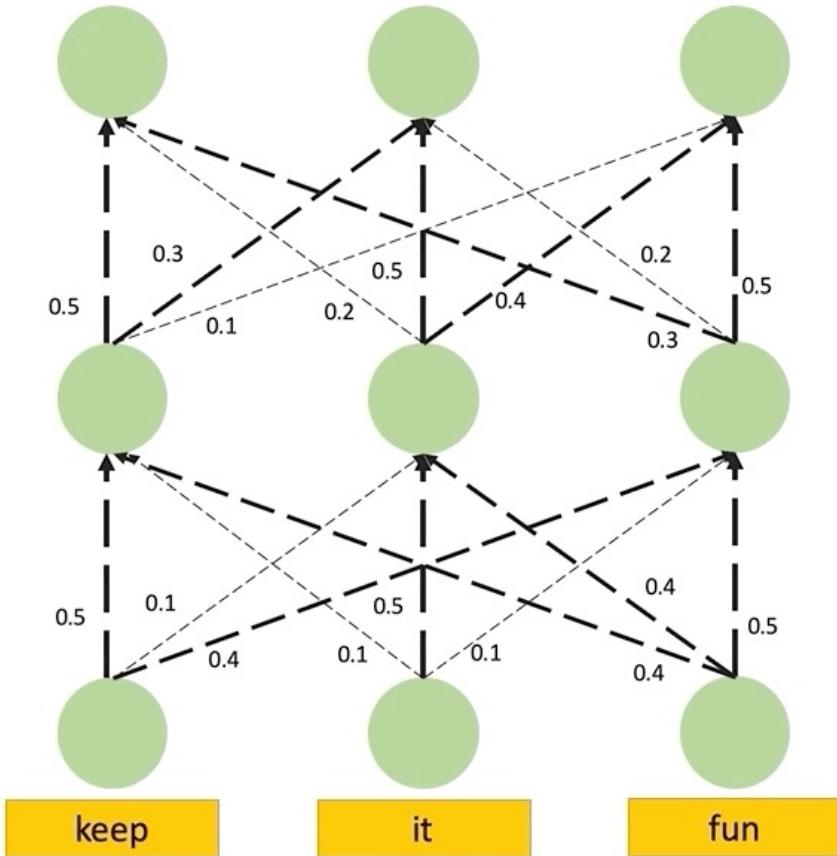
- Path as a proportion of information transferred between nodes (multiply the weights of all edges)
- More than one path possible, the total amount of information is the sum of all possible paths
- Recursive formula
(l_k - k-th layer, $j \leq i$, A – raw attention):
$$\tilde{A}(l_i) = \begin{cases} A(l_i)\tilde{A}(l_{i-1}) & \text{if } i > j \\ A(l_i) & \text{if } i = k \end{cases}$$
- Polynomial time complexity $O(d * n^2)$



Source: [Quantifying Attention Flow in Transformers | Samira Abnar](#)

Attention flow

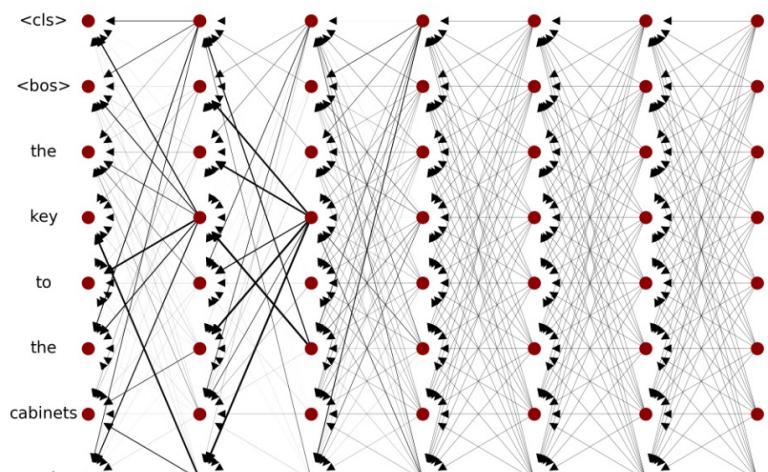
- Treating attention graph as flow network where capacities of edges are attention weights
- We calculate maximum attention flow from any node in any of the layers to any of the input nodes
- Solvable using any maximum flow algorithm e.g. Ford–Fulkerson algorithm
- Compared to attention rollout, the weight of single path is the minimum value of the weights, instead of the product
- Polynomial time complexity $O(d * n^4)$



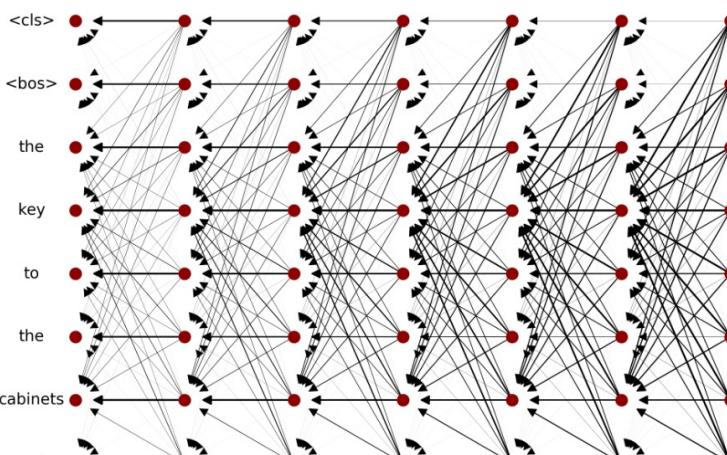
Source: [Quantifying Attention Flow in Transformers | Samira Abnar](#)

Comparison

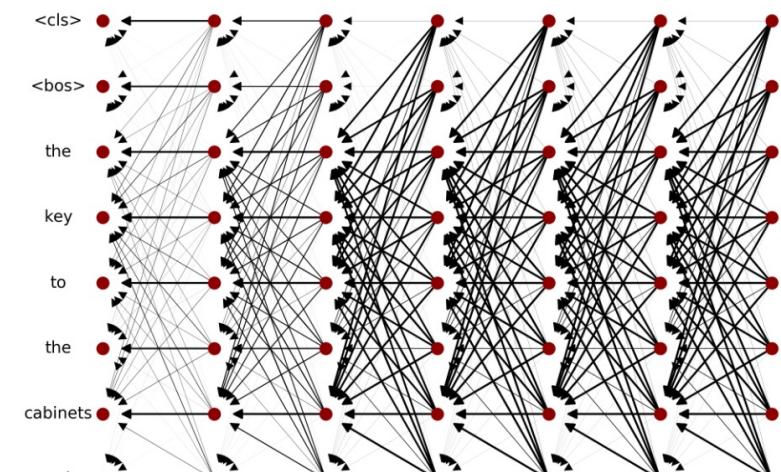
Visualization of correctly classified example.



(a) Embedding attentions



(b) Attention rollout



(c) Attention flow

Comparison cont.

The first and second sentences were classified correctly, while 3rd was misclassified.

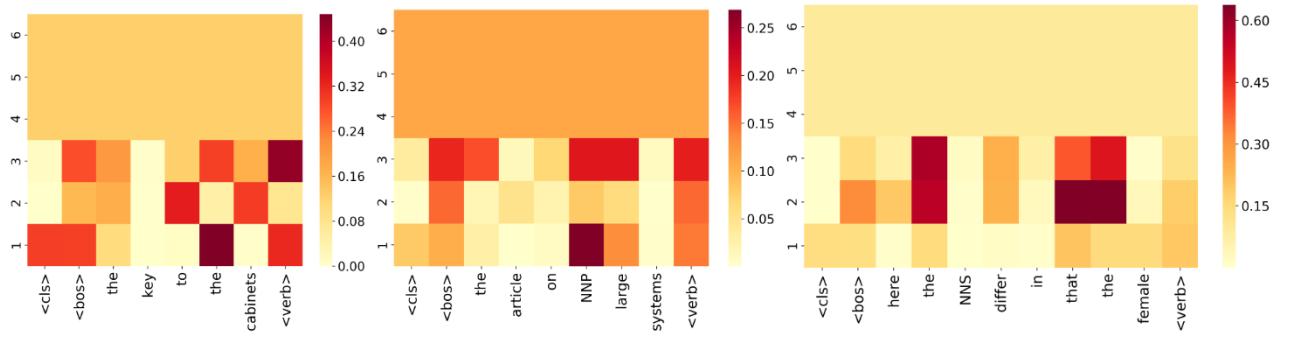


Figure 2: Raw Attention maps for the CLS token at different layers.

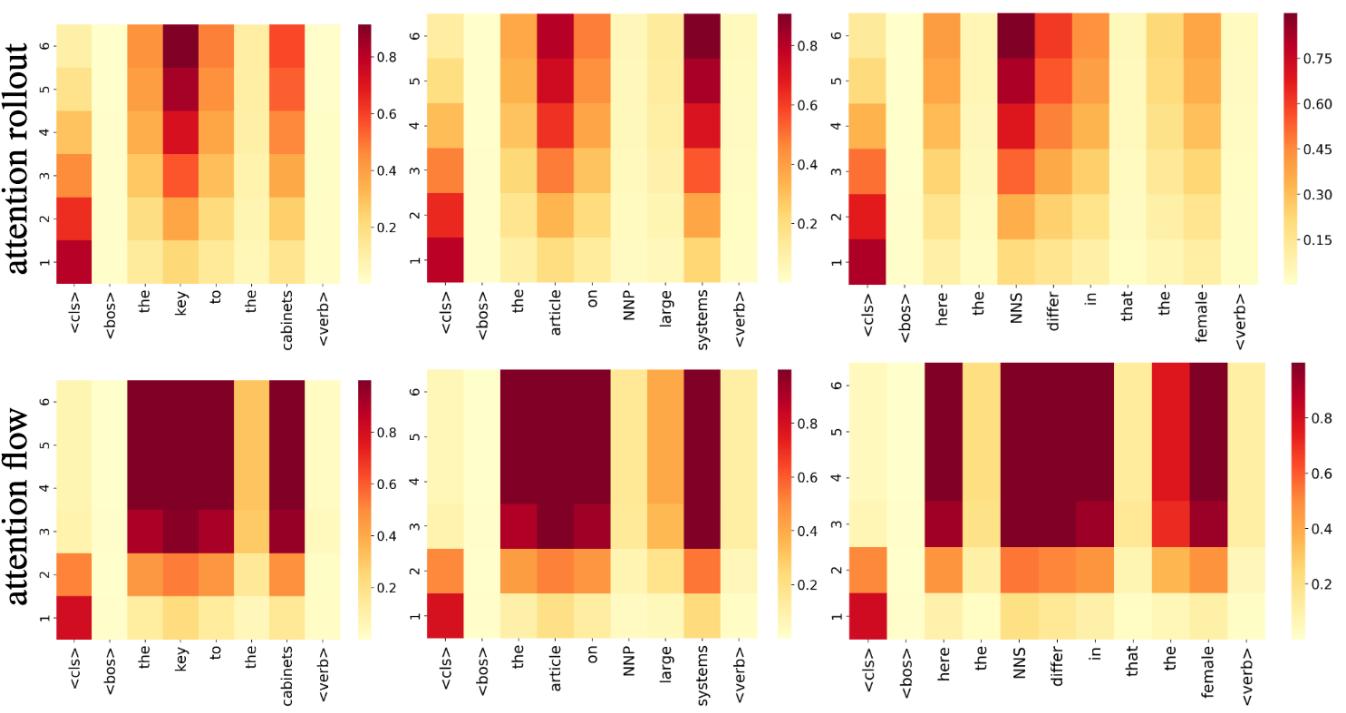


Figure 3: Attention maps for the CLS token

Results

SpearmanR correlation measures rank correlation, i.e., assesses how well the relationship between two variables can be described using a monotonic function.

	L1	L2	L3	L4	L5	L6
Raw	0.69±0.27	0.10±0.43	-0.11±0.49	-0.09±0.52	0.20±0.45	0.29±0.39
Rollout	0.32±0.26	0.38±0.27	0.51±0.26	0.62±0.26	0.70±0.25	0.71±0.24
Flow	0.32±0.26	0.44±0.29	0.70±0.25	0.70±0.22	0.71±0.22	0.70±0.22

Table 1: SpearmanR correlation of attention based importance with blank-out scores for 2000 samples from the test set for the verb number prediction model.

	L1	L2	L3	L4	L5	L6
Raw	0.53±0.33	0.16±0.38	-0.06±0.42	0.00±0.47	0.24±0.40	0.46±0.35
Rollout	0.22±0.31	0.27±0.32	0.39±0.32	0.47±0.32	0.53±0.32	0.54±0.31
Flow	0.22±0.31	0.31±0.34	0.54±0.32	0.61±0.28	0.60±0.28	0.61±0.28

Table 2: SpearmanR correlation of attention based importance with input gradients for 2000 samples from the test set for the verb number prediction model.

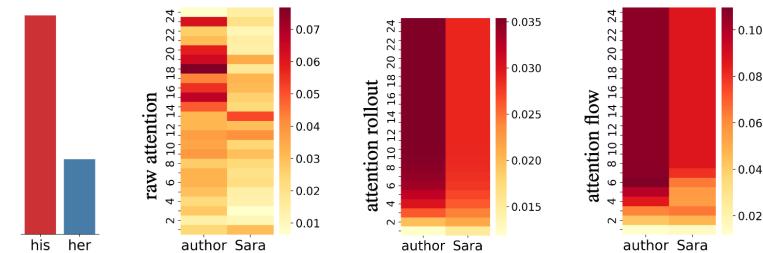
Results (other task)

Result from the sentiment analysis task.

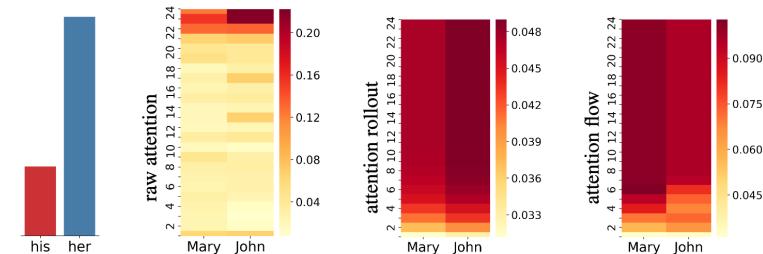
	L1	L3	L5	L6
Raw	0.12 ± 0.21	0.09 ± 0.21	0.08 ± 0.20	0.09 ± 0.21
Rollout	0.11 ± 0.19	0.12 ± 0.21	0.13 ± 0.21	0.13 ± 0.20
Flow	0.11 ± 0.19	0.11 ± 0.21	0.12 ± 0.22	0.14 ± 0.21

Table 3: SpearmanR correlation of attention based importance with input gradients for 100 samples from the test set for the DistillBERT model fine tuned on SST-2.

Result from the pre-trained Bert
(resolving pronounce in sentence).



(a) “The author talked to Sara about *mask* book”



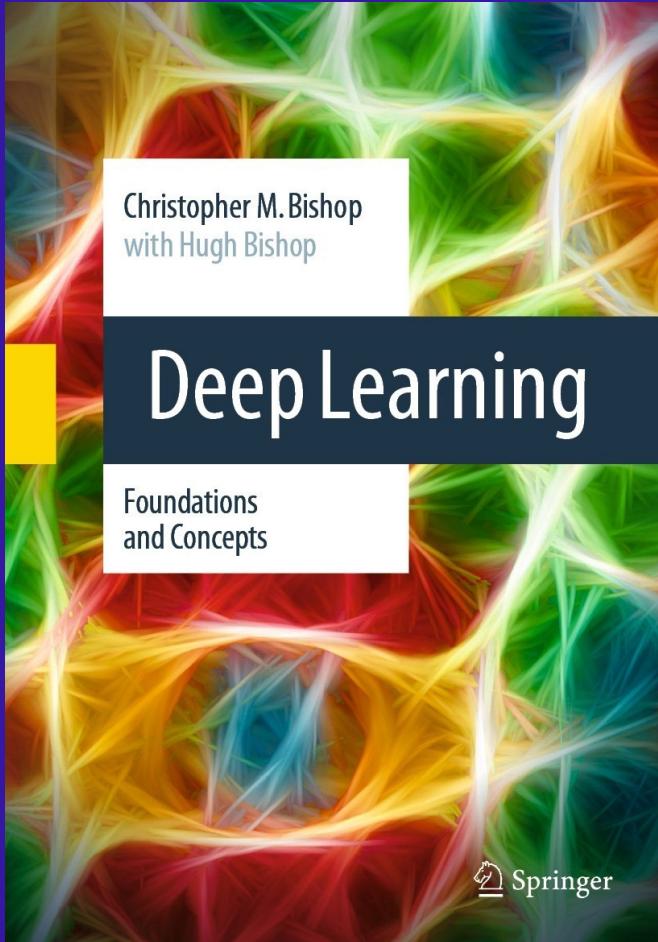
(b) “*Mary* convinced *John* of *mask* *love*”

Figure 4: Bert attention maps. We look at the attention weights from the mask embedding to the two potential references for it, e.g. “author” and “Sara” in (a) and “Mary” and “John” in (b). The bars, at the left, show the relative predicted probability for the two possible pronouns, “his” and “her”.

Conclusions

- Proposed methods provide results more correlated with blank-out scores and input gradients than raw attention.
- Attention rollout provides more strict patterns (but not necessarily more accurate results), while attention flow is more flexible.
- Those methods are easy to implement and task-agnostic.
- In paper applied for encoder-only. To apply for decoder, normalization is needed since both attention rollout and attention flow will be biased toward the input sentence.
- It is possible to analyze each head of attention separately (*).
- As with many other XAI methods, there is a high emphasis that it does not necessarily identify hidden embeddings

Review of Deep Learning: Foundations and Concepts



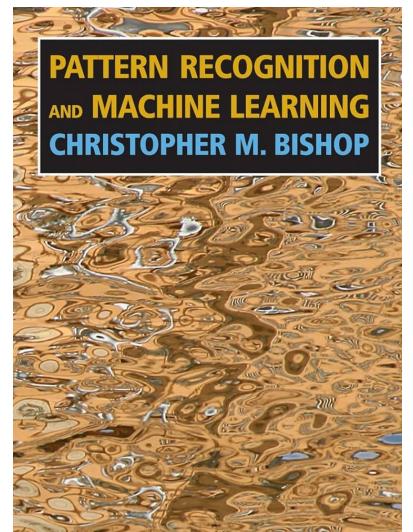
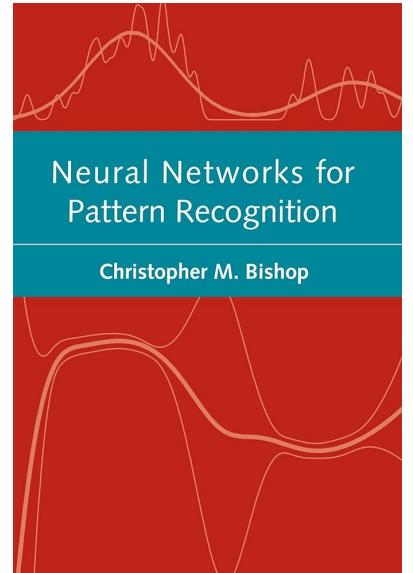
Authors



Chris Bishop is a Technical Fellow at Microsoft and is the Director of Microsoft Research AI4Science. Professor of Computer Science, teaching at the University of Edinburgh. Author of two other influential ML textbooks: **Neural Networks for Pattern Recognition** and **Pattern Recognition and Machine Learning**.



Hugo Bishop is an Applied Scientist at Wayve, a deep learning autonomous driving company in London where he designs and trains deep neural networks. Cambridge graduate in Machine Learning and Machine Intelligence.



Goals of the book

- Provide a solid basis for deep learning, both for newcomers and those already experienced in the field
- Focus on critical ideas and foundations
- Focus on technicals only, without discussing ethical, societal, or legal aspects
- No literature review due to rapid changes
- Suited for two-semester undergraduate or postgraduate ML course

Book structure

Preface	v	
Contents	xi	
1 The Deep Learning Revolution	1	
1.1 The Impact of Deep Learning	2	
1.1.1 Medical diagnosis	2	
1.1.2 Protein structure	3	
1.1.3 Image synthesis	4	
1.1.4 Large language models	5	
1.2 A Tutorial Example	6	
1.2.1 Synthetic data	6	
1.2.2 Linear models	8	
1.2.3 Error function	8	
1.2.4 Model complexity	9	
1.2.5 Regularization	12	
1.2.6 Model selection	14	
1.3 A Brief History of Machine Learning	16	
1.3.1 Single-layer networks	17	
1.3.2 Backpropagation	18	
1.3.3 Deep networks	20	
2 Probabilities	23	
2.1 The Rules of Probability	25	
2.1.1 A medical screening example	25	
2.1.2 The sum and product rules	26	
2.1.3 Bayes' theorem	28	
2.1.4 Medical screening revisited	30	
2.1.5 Prior and posterior probabilities	31	
2.2 Probability Densities	31	
2.2.1 Example distributions	32	
2.2.2 Expectations and covariances	33	
2.3 The Gaussian Distribution	34	
2.3.1 Mean and variance	36	
2.3.2 Likelihood function	37	
2.3.3 Bias of maximum likelihood	39	
2.3.4 Linear regression	40	
2.4 Transformation of Densities	42	
2.4.1 Multivariate distributions	44	
2.5 Information Theory	46	
2.5.1 Entropy	46	
2.5.2 Physics perspective	47	
2.5.3 Differential entropy	49	
2.5.4 Maximum entropy	50	
2.5.5 Kullback–Leibler divergence	51	
2.5.6 Conditional entropy	53	
2.5.7 Mutual information	54	
2.6 Bayesian Probabilities	54	
2.6.1 Model parameters	55	
2.6.2 Regularization	56	
2.6.3 Bayesian machine learning	57	
Exercises	58	
3 Standard Distributions	65	
3.1 Discrete Variables	66	
3.1.1 Bernoulli distribution	66	
3.1.2 Binomial distribution	67	
3.1.3 Multinomial distribution	68	
3.2 The Multivariate Gaussian	70	
3.2.1 Geometry of the Gaussian	71	
3.2.2 Moments	74	
3.2.3 Limitations	75	
3.2.4 Conditional distribution	76	
3.2.5 Marginal distribution	79	
3.2.6 Bayes' theorem	81	
3.2.7 Maximum likelihood	84	
3.2.8 Sequential estimation	85	
3.2.9 Mixtures of Gaussians	86	
3.3 Periodic Variables	89	
3.3.1 Von Mises distribution	89	
3.4 The Exponential Family	94	
3.4.1 Sufficient statistics	97	
3.5 Nonparametric Methods	98	

3.5.1	Histograms	98
3.5.2	Kernel densities	100
3.5.3	Nearest-neighbours	103
	Exercises	105
4	Single-layer Networks: Regression	111
4.1	Linear Regression	112
4.1.1	Basis functions	112
4.1.2	Likelihood function	114
4.1.3	Maximum likelihood	115
4.1.4	Geometry of least squares	117
4.1.5	Sequential learning	117
4.1.6	Regularized least squares	118
4.1.7	Multiple outputs	119
4.2	Decision theory	120
4.3	The Bias–Variance Trade-off	123
	Exercises	128
5	Single-layer Networks: Classification	131
5.1	Discriminant Functions	132
5.1.1	Two classes	132
5.1.2	Multiple classes	134
5.1.3	1-of- K coding	135
5.1.4	Least squares for classification	136
5.2	Decision Theory	138
5.2.1	Misclassification rate	139
5.2.2	Expected loss	140
5.2.3	The reject option	142
5.2.4	Inference and decision	143
5.2.5	Classifier accuracy	147
5.2.6	ROC curve	148
5.3	Generative Classifiers	150
5.3.1	Continuous inputs	152
5.3.2	Maximum likelihood solution	153
5.3.3	Discrete features	156
5.3.4	Exponential family	156
5.4	Discriminative Classifiers	157
5.4.1	Activation functions	158
5.4.2	Fixed basis functions	158
5.4.3	Logistic regression	159
5.4.4	Multi-class logistic regression	161
5.4.5	Probit regression	163
5.4.6	Canonical link functions	164
	Exercises	166
6	Deep Neural Networks	171
6.1	Limitations of Fixed Basis Functions	172
6.1.1	The curse of dimensionality	172
6.1.2	High-dimensional spaces	175
6.1.3	Data manifolds	176
6.1.4	Data-dependent basis functions	178
6.2	Multilayer Networks	180
6.2.1	Parameter matrices	181
6.2.2	Universal approximation	181
6.2.3	Hidden unit activation functions	182
6.2.4	Weight-space symmetries	185
6.3	Deep Networks	186
6.3.1	Hierarchical representations	187
6.3.2	Distributed representations	187
6.3.3	Representation learning	188
6.3.4	Transfer learning	189
6.3.5	Contrastive learning	191
6.3.6	General network architectures	193
6.3.7	Tensors	194
6.4	Error Functions	194
6.4.1	Regression	194
6.4.2	Binary classification	196
6.4.3	multiclass classification	197
6.5	Mixture Density Networks	198
6.5.1	Robot kinematics example	198
6.5.2	Conditional mixture distributions	199
6.5.3	Gradient optimization	201
6.5.4	Predictive distribution	202
	Exercises	204
7	Gradient Descent	209
7.1	Error Surfaces	210
7.1.1	Local quadratic approximation	211
7.2	Gradient Descent Optimization	213
7.2.1	Use of gradient information	214
7.2.2	Batch gradient descent	214
7.2.3	Stochastic gradient descent	214
7.2.4	Mini-batches	216
7.2.5	Parameter initialization	216
7.3	Convergence	218
7.3.1	Momentum	220
7.3.2	Learning rate schedule	222
7.3.3	RMSProp and Adam	223
7.4	Normalization	224
7.4.1	Data normalization	226

7.4.2	Batch normalization	227
7.4.3	Layer normalization	229
Exercises		230
8	Backpropagation	233
8.1	Evaluation of Gradients	234
8.1.1	Single-layer networks	234
8.1.2	General feed-forward networks	235
8.1.3	A simple example	238
8.1.4	Numerical differentiation	239
8.1.5	The Jacobian matrix	240
8.1.6	The Hessian matrix	242
8.2	Automatic Differentiation	244
8.2.1	Forward-mode automatic differentiation	246
8.2.2	Reverse-mode automatic differentiation	249
Exercises		250
9	Regularization	253
9.1	Inductive Bias	254
9.1.1	Inverse problems	254
9.1.2	No free lunch theorem	255
9.1.3	Symmetry and invariance	256
9.1.4	Equivariance	259
9.2	Weight Decay	260
9.2.1	Consistent regularizers	262
9.2.2	Generalized weight decay	264
9.3	Learning Curves	266
9.3.1	Early stopping	266
9.3.2	Double descent	268
9.4	Parameter Sharing	270
9.4.1	Soft weight sharing	271
9.5	Residual Connections	274
9.6	Model Averaging	277
9.6.1	Dropout	279
Exercises		281
10	Convolutional Networks	287
10.1	Computer Vision	288
10.1.1	Image data	289
10.2	Convolutional Filters	290
10.2.1	Feature detectors	290
10.2.2	Translation equivariance	291
10.2.3	Padding	294
10.2.4	Strided convolutions	294
10.2.5	Multi-dimensional convolutions	295
10.2.6	Pooling	296
10.2.7	Multilayer convolutions	298
10.2.8	Example network architectures	299
10.3	Visualizing Trained CNNs	302
10.3.1	Visual cortex	302
10.3.2	Visualizing trained filters	303
10.3.3	Saliency maps	305
10.3.4	Adversarial attacks	306
10.3.5	Synthetic images	308
10.4	Object Detection	308
10.4.1	Bounding boxes	309
10.4.2	Intersection-over-union	310
10.4.3	Sliding windows	311
10.4.4	Detection across scales	313
10.4.5	Non-max suppression	314
10.4.6	Fast region CNNs	314
10.5	Image Segmentation	315
10.5.1	Convolutional segmentation	315
10.5.2	Up-sampling	316
10.5.3	Fully convolutional networks	318
10.5.4	The U-net architecture	319
10.6	Style Transfer	320
Exercises		322
11	Structured Distributions	325
11.1	Graphical Models	326
11.1.1	Directed graphs	326
11.1.2	Factorization	327
11.1.3	Discrete variables	329
11.1.4	Gaussian variables	332
11.1.5	Binary classifier	334
11.1.6	Parameters and observations	334
11.1.7	Bayes' theorem	336
11.2	Conditional Independence	337
11.2.1	Three example graphs	338
11.2.2	Explaining away	341
11.2.3	D-separation	343
11.2.4	Naive Bayes	344
11.2.5	Generative models	346
11.2.6	Markov blanket	347
11.2.7	Graphs as filters	348
11.3	Sequence Models	349
11.3.1	Hidden variables	352
Exercises		353

12 Transformers	357
12.1 Attention	358
12.1.1 Transformer processing	360
12.1.2 Attention coefficients	361
12.1.3 Self-attention	362
12.1.4 Network parameters	363
12.1.5 Scaled self-attention	366
12.1.6 Multi-head attention	366
12.1.7 Transformer layers	368
12.1.8 Computational complexity	370
12.1.9 Positional encoding	371
12.2 Natural Language	374
12.2.1 Word embedding	375
12.2.2 Tokenization	377
12.2.3 Bag of words	378
12.2.4 Autoregressive models	379
12.2.5 Recurrent neural networks	380
12.2.6 Backpropagation through time	381
12.3 Transformer Language Models	382
12.3.1 Decoder transformers	383
12.3.2 Sampling strategies	386
12.3.3 Encoder transformers	388
12.3.4 Sequence-to-sequence transformers	390
12.3.5 Large language models	390
12.4 Multimodal Transformers	394
12.4.1 Vision transformers	395
12.4.2 Generative image transformers	396
12.4.3 Audio data	399
12.4.4 Text-to-speech	400
12.4.5 Vision and language transformers	402
Exercises	403
13 Graph Neural Networks	407
13.1 Machine Learning on Graphs	409
13.1.1 Graph properties	410
13.1.2 Adjacency matrix	410
13.1.3 Permutation equivariance	411
13.2 Neural Message-Passing	412
13.2.1 Convolutional filters	413
13.2.2 Graph convolutional networks	414
13.2.3 Aggregation operators	416
13.2.4 Update operators	418
13.2.5 Node classification	419
13.2.6 Edge classification	420
13.2.7 Graph classification	420
Exercises	420
13.3 General Graph Networks	420
13.3.1 Graph attention networks	421
13.3.2 Edge embeddings	421
13.3.3 Graph embeddings	422
13.3.4 Over-smoothing	422
13.3.5 Regularization	423
13.3.6 Geometric deep learning	424
Exercises	425
14 Sampling	429
14.1 Basic Sampling Algorithms	430
14.1.1 Expectations	430
14.1.2 Standard distributions	431
14.1.3 Rejection sampling	433
14.1.4 Adaptive rejection sampling	435
14.1.5 Importance sampling	437
14.1.6 Sampling-importance-resampling	439
14.2 Markov Chain Monte Carlo	440
14.2.1 The Metropolis algorithm	441
14.2.2 Markov chains	442
14.2.3 The Metropolis–Hastings algorithm	445
14.2.4 Gibbs sampling	446
14.2.5 Ancestral sampling	450
14.3 Langevin Sampling	451
14.3.1 Energy-based models	452
14.3.2 Maximizing the likelihood	453
14.3.3 Langevin dynamics	454
Exercises	456
15 Discrete Latent Variables	459
15.1 K-means Clustering	460
15.1.1 Image segmentation	464
15.2 Mixtures of Gaussians	466
15.2.1 Likelihood function	468
15.2.2 Maximum likelihood	470
15.3 Expectation–Maximization Algorithm	474
15.3.1 Gaussian mixtures	478
15.3.2 Relation to K-means	480
15.3.3 Mixtures of Bernoulli distributions	481
15.4 Evidence Lower Bound	485
15.4.1 EM revisited	486
15.4.2 Independent and identically distributed data	488
15.4.3 Parameter priors	489
15.4.4 Generalized EM	489
15.4.5 Sequential EM	490
Exercises	490

16 Continuous Latent Variables	495	19 Autoencoders	563
16.1 Principal Component Analysis	497	19.1 Deterministic Autoencoders	564
16.1.1 Maximum variance formulation	497	19.1.1 Linear autoencoders	564
16.1.2 Minimum-error formulation	499	19.1.2 Deep autoencoders	565
16.1.3 Data compression	501	19.1.3 Sparse autoencoders	566
16.1.4 Data whitening	502	19.1.4 Denoising autoencoders	567
16.1.5 High-dimensional data	504	19.1.5 Masked autoencoders	567
16.2 Probabilistic Latent Variables	506	19.2 Variational Autoencoders	569
16.2.1 Generative model	506	19.2.1 Amortized inference	572
16.2.2 Likelihood function	507	19.2.2 The reparameterization trick	574
16.2.3 Maximum likelihood	509	Exercises	578
16.2.4 Factor analysis	513		
16.2.5 Independent component analysis	514		
16.2.6 Kalman filters	515		
16.3 Evidence Lower Bound	516		
16.3.1 Expectation maximization	518		
16.3.2 EM for PCA	519		
16.3.3 EM for factor analysis	520		
16.4 Nonlinear Latent Variable Models	522		
16.4.1 Nonlinear manifolds	522		
16.4.2 Likelihood function	524		
16.4.3 Discrete data	526		
16.4.4 Four approaches to generative modelling	527		
Exercises	527		
17 Generative Adversarial Networks	533		
17.1 Adversarial Training	534		
17.1.1 Loss function	535		
17.1.2 GAN training in practice	536		
17.2 Image GANs	539		
17.2.1 CycleGAN	539		
Exercises	544		
18 Normalizing Flows	547		
18.1 Coupling Flows	549		
18.2 Autoregressive Flows	552		
18.3 Continuous Flows	554		
18.3.1 Neural differential equations	554		
18.3.2 Neural ODE backpropagation	555		
18.3.3 Neural ODE flows	557		
Exercises	559		
19 Autoencoders	563		
19.1 Deterministic Autoencoders	564		
19.1.1 Linear autoencoders	564		
19.1.2 Deep autoencoders	565		
19.1.3 Sparse autoencoders	566		
19.1.4 Denoising autoencoders	567		
19.1.5 Masked autoencoders	567		
19.2 Variational Autoencoders	569		
19.2.1 Amortized inference	572		
19.2.2 The reparameterization trick	574		
Exercises	578		
20 Diffusion Models	581		
20.1 Forward Encoder	582		
20.1.1 Diffusion kernel	583		
20.1.2 Conditional distribution	584		
20.2 Reverse Decoder	585		
20.2.1 Training the decoder	587		
20.2.2 Evidence lower bound	588		
20.2.3 Rewriting the ELBO	589		
20.2.4 Predicting the noise	591		
20.2.5 Generating new samples	592		
20.3 Score Matching	594		
20.3.1 Score loss function	595		
20.3.2 Modified score loss	596		
20.3.3 Noise variance	597		
20.3.4 Stochastic differential equations	598		
20.4 Guided Diffusion	599		
20.4.1 Classifier guidance	600		
20.4.2 Classifier-free guidance	600		
Exercises	603		
Appendix A Linear Algebra	609		
A.1 Matrix Identities	609		
A.2 Traces and Determinants	610		
A.3 Matrix Derivatives	611		
A.4 Eigenvectors	612		
Appendix B Calculus of Variations	617		
Appendix C Lagrange Multipliers	621		
Bibliography	625		
Index	641		

Review (pros)

- It starts with the basics: probability, distributions, regression, etc.; the only prerequisite seems to be calculus and linear algebra
- Provides intuitive and easy-to-understand examples and analogies
- Simple (for an academic book) language used
- Lots of figures
- Most of the first-time appearing keywords provided with a comprehensive explanation
- **Available online with a complete set of figures available for download**
- Equiped with the exerclies ...

Exercises sneak peak

10.1 (*) Consider a fixed weight vector \mathbf{w} and show that the input vector \mathbf{x} that maximizes the scalar product $\mathbf{w}^T \mathbf{x}$, subject to the constraint that $\|\mathbf{x}\|^2$ is constant, is given by $\mathbf{x} = \alpha \mathbf{w}$ for some scalar α . This can most easily be done using a Lagrange multiplier.

10.2 (**) Consider a convolutional network layer with a one-dimensional input array and a one-dimensional feature map as shown in Figure 10.2, in which the input array has dimensionality 5 and the filters have width 3 with a stride of 1. Show that this can be expressed as a special case of a fully connected layer by writing down the weight matrix in which missing connections are replaced by zeros and where shared parameters are indicated by using replicated entries. Ignore any bias parameters.

10.3 (*) Explicitly calculate the output of the following convolution of a 4×4 input matrix with a 2×2 filter:

$$\begin{array}{|c|c|c|c|} \hline 2 & 5 & -3 & 0 \\ \hline 0 & 6 & 0 & -4 \\ \hline -1 & -3 & 0 & 2 \\ \hline 5 & 0 & 0 & 3 \\ \hline \end{array} * \begin{array}{|c|c|} \hline -2 & 0 \\ \hline 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline ? & ? & ? \\ \hline ? & ? & ? \\ \hline ? & ? & ? \\ \hline \end{array} \quad (10.18)$$

10.4 (**) If an image \mathbf{I} has $J \times K$ pixels and a filter \mathbf{K} has $L \times M$ elements, write down the limits for the two summations in (10.2). In the mathematics literature, the operation (10.2) would be called a *cross-correlation*, whereas a *convolution* would be defined by

$$C(j, k) = \sum_l \sum_m I(j-l, k-m) K(l, m). \quad (10.19)$$

Write down the limits for the summations in (10.19). Show that (10.19) can be written in the equivalent ‘flipped’ form

$$C(j, k) = \sum_l \sum_m I(j+l, k+m) K(l, m) \quad (10.20)$$

and again write down the limits for the summations.

10.5 (*) In mathematics, a convolution for a continuous variable x is defined by

$$F(x) = \int_{-\infty}^{\infty} G(y) k(x-y) dy \quad (10.21)$$

where $k(x-y)$ is the kernel function. By considering a discrete approximation to the integral, explain the relationship to a convolutional layer, defined by (10.19), in a CNN.

Review (cons)

- ... but they are purely mathematical
- Thus, as ML textbook lacks a practical part (coding and working with real data)
- Despite the name, the authors introduce this as an ML textbook (in the Preface), yet it lacks a few basic ML concepts, e.g., decision tree, SVM, ensemble, and reinforcement learning...
- Relatively not approachable by those without (mathematical) academic background (?)

Thank You for your attention!

Filip Kołodziejczyk



MI2.AI Seminar, Warsaw, May 13th, 2024