Deep Learning

Class IV: More Transformers

Hédi Hadiji

Université Paris-Saclay - CentraleSupelec

hedi.hadiji@l2s.centralesupelec.fr

October, 2025

Table of Contents

Last time

Last time:

- Attention
- Transformers

Today: More transformers

Overview

- Recap of previous lecture: attention, normalization, residuals, autoregressive prediction.
- Today's goals:
  - Understand how Transformers handle input–output pairs.
  - Explore encoder–decoder structure.
  - Compare model families (GPT, BERT, T5).
  - See how attention generalizes to vision.

Last time

(Self-)Attention

Given a sequence of embeddings, produce a new sequence of embeddings that accounts for contextual information from other tokens.

(Decoder-only) Transformers and Auto-regressive prediction

Take a data point $[o_1, o_2, \ldots, o_{T-1}, o_T]$.
Give input sequence $[o_1, o_2, \ldots, o_{T-1}]$ try to predict target $[o_2, \ldots, o_{T-1}, o_T]$

Use **positional encoding** to incorporate token positions + **causal masking** to force the model to only predict tokens from past tokens.

(Causal) (Self-)Attention

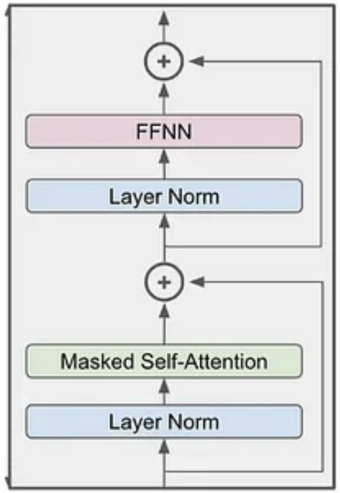
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

K: keys, Q: queries, V: values

	K_1	K_2	K_3
Q_1	$w_1^{(1)}$	0	0
Q_2	$w_1^{(2)}$	$w_2^{(2)}$	0
Q_3	$w_1^{(3)}$	$w_2^{(3)}$	$w_3^{(3)}$

$$\times \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix}$$

Generative Pretrained Transformers



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.

Training takes massive engineering to handle this scale. GPT-3 likely trained 1024 V100 GPUs for about a month.

Recap: Autoregressive Transformers

Decoder-only computes

$$p(o_{t+1}|o_1, \ldots, o_t)$$

in parallel for all elements of the sequence.

Question

How to use this for translation? For summarization? We want to be able to condition generation on different input.

I.e. have a next-token generation which can condition on different data

$$p(o_{t+1}|o_1, \ldots, o_t, c_1, \ldots, c_{T_c})$$

where $(c_1, \ldots, c_{T_c})$ is contextual data (text to summarize or translate).

Table of Contents

Sequence-to-Sequence (Encoder–Decoder)

Goal

Map a source sequence (context) to a distinct target sequence (e.g., translation, summarization).

Example data

Source (English):
`Bob ate the red apple.`

Target (French):
`Bob a mangé la pomme rouge.`

Training target

Decoder input (shifted):
`<s> Bob a mangé la pomme rouge`
Predictions: $\widehat{y}_1, \widehat{y}_2, \ldots$ compared to ground-truth tokens.

Handmade solution

Train our previous model on the sequences of the form

`'<fr>Comment vas-tu?</fr> <en>How are you?</en>'`

Then auto-regressive prediction might learn to predict a translation.

There is a more structured way to represent the contextual data.

Cross-Attention

Cross-Attention

Given a sequence of context embeddings (encoder outputs) and a sequence of input embeddings (decoder states).

- Queries (Q): from the input (decoder)
- Keys (K), Values (V): from the context (encoder)

$$\text{CrossAtt}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V$$

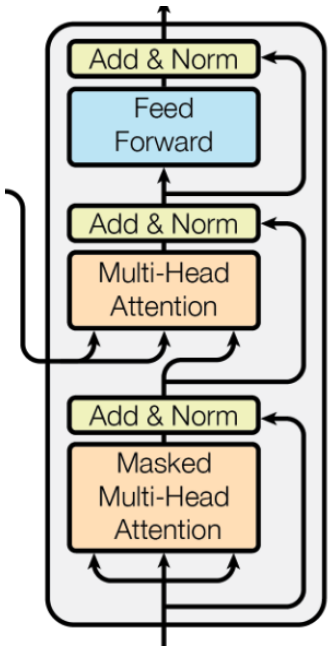
	K_1	K_2	K_3	K_4
Q_1	$w_1^{(1)}$	$w_2^{(1)}$	$w_3^{(1)}$	$w_4^{(1)}$
Q_2	$w_1^{(2)}$	$w_2^{(2)}$	$w_3^{(2)}$	$w_4^{(2)}$
Q_3	$w_1^{(3)}$	$w_2^{(3)}$	$w_3^{(3)}$	$w_4^{(1)}$

$\times$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix}$$

Cross-attention produces a new sequence of input-aware embeddings by letting the input queries attend to keys/values coming from the context.

Decoder Block



Two types of argument:

- the output sequence embeddings
- the context embeddings

Like decoder-only architecture, but add **queries** and **values** from an encoder.

Encoder-Decoder Architecture

- **Encoder:** unmasked self-attention over input. Uses attention mechanism to incorporate all the contextual information in the embeddings.
- **Decoder:** masked self-attention over output + cross-attention to encoder outputs.

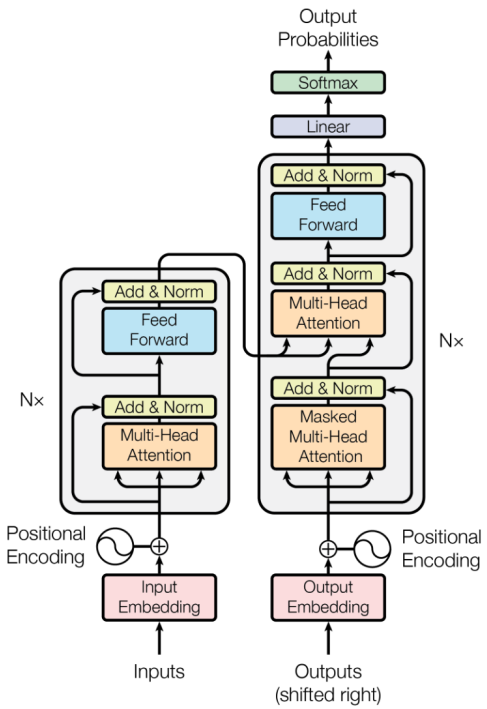


Figure 1: The Transformer - model architecture.

A pipeline for translation

Goal: have an English to French translator

Dataset: Pairs of sentences : French-English

Complete trained model will have

- context: English sentence
- input : incomplete French sentence
- output : Probabilities of next token in the French sentence

Training

Example

Context: `Bob ate a green apple.`
Target: `Bob a mangé une pomme verte.`

Encoder

Context (with positional encodings) $\rightarrow$ encoder outputs $(c_1, \ldots, c_{T_c})$.

What we train

Feed decoder with the (shifted) target prefix and predict the next token at each step.

Decoder (masked + cross-attn)

Decoder input: $(o_0, o_1, \ldots, o_{T_o-1})$ (positional enc. + causal mask)

Predictions: $(\widehat{o}_1, \widehat{o}_2, \ldots, \widehat{o}_{T_o})$, where each $\widehat{o}_t$ is conditioned on $o_{<t}$ and $c_{1:T_c}$.

Teacher forcing

During training the decoder always receives the ground-truth prefix

Training objective: next-token cross-entropy

$$\mathcal{L} = -\sum_{t=1}^{T_o} \log p\big(o_t \mid o_{<t},\ c_{1:T_c}\big).$$

Table of Contents

Transformers as an architecture piece

We have seen:

- Decoder-only : for text-generation (GPT-3)
- Encoder-Decoder: for translation/summarization

Many other modern architectures are based on transformers.

Pre-training and Finetuning Models

Idea: Start from a model pretrained on a large dataset and adapt it to a specific downstream task.

Why it works:

- Pretraining captures general representations (syntax, semantics, visual features…)
- Fine-tuning transfers this knowledge with fewer task-specific examples

Procedure:

1. Load pretrained backbone
2. Replace or add output layers / heads for the target task
3. Train with smaller learning rate to preserve pretrained features

Variants:

- *Full fine-tuning:* Update all parameters
- *Partial fine-tuning:* Freeze backbone, train only classifier head
- *Adapters / LoRA:* Insert lightweight layers for efficient adaptation

BERT: Bidirectional Encoder Representations from Transformers (2018)

Main idea:

- Pre-train deep bidirectional representations using the transformer **encoder**.
- Use a large text corpus (Wikipedia + BookCorpus) to learn contextual embeddings.
- **Fine-tune** on downstream tasks with minimal architecture changes.

Architecture:

- Transformer **encoder-only** (no decoder).
- Each token attends to all others on both sides (left + right context).
- Input = `[CLS]` + sentence A + `[SEP]` + sentence B.
- Positional embeddings + token embeddings + segment embeddings.

BERT:

Pre-training objectives:

- **Masked Language Modeling (MLM):** Randomly mask 15% of tokens and predict them using context.
- **Next Sentence Prediction (NSP):** Classify whether sentence B follows sentence A.

Fine-tuning:

- Add a simple task-specific head (e.g. classification, QA, NER).
- Update all model parameters jointly.

Key variants: $\text{BERT}_{\text{BASE}}$ (12 layers, 110M params), $\text{BERT}_{\text{LARGE}}$ (24 layers, 340M params)
see the github repo and the paper .

Masking: which tokens attend to which

Component	Masked?	Attends to
Encoder self-attn	No	all input tokens
Decoder self-attn	Yes	previous output tokens only
Decoder cross-attn	No	all encoder outputs

- Training: teacher forcing with parallel input/output pairs.
- Loss: next-token cross-entropy on output vocabulary.
- Typically same d_{model} and positional encoding for both sides.

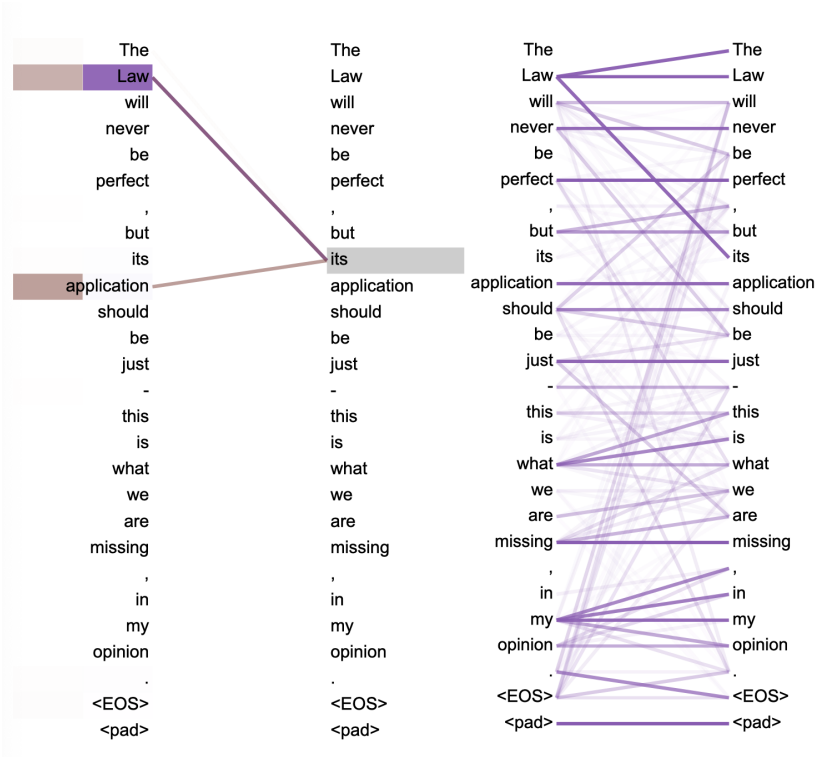
Transformer Families

Model	Architecture	Objective	Typical Tasks
GPT	Decoder-only	Next-token prediction	Text generation
BERT	Encoder-only	Masked LM (autoencoding)	Classification, QA
T5 / Transformer	Encoder–Decoder	Seq2Seq prediction	Translation, summarization

- Encoder-only models read
- Decoder-only models write
- Encoder-Decoder models translate

Visualizing Attention

Like in CNNs we can visualize what is computed by a model by extracting the attention scores for given inputs, at a given layer and a given head.

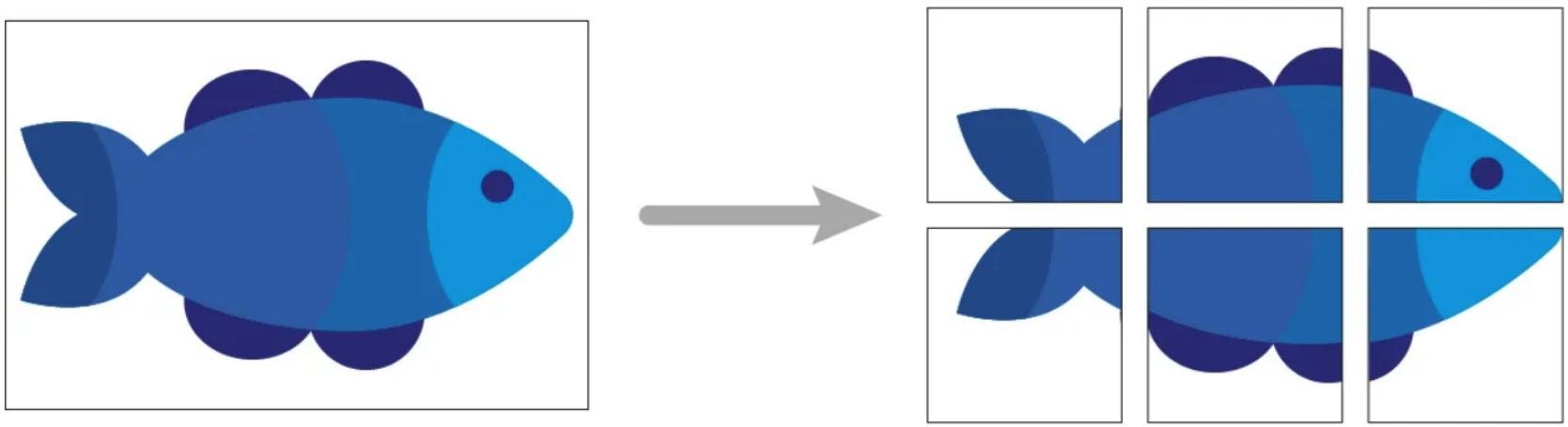


see this demo.

Table of Contents

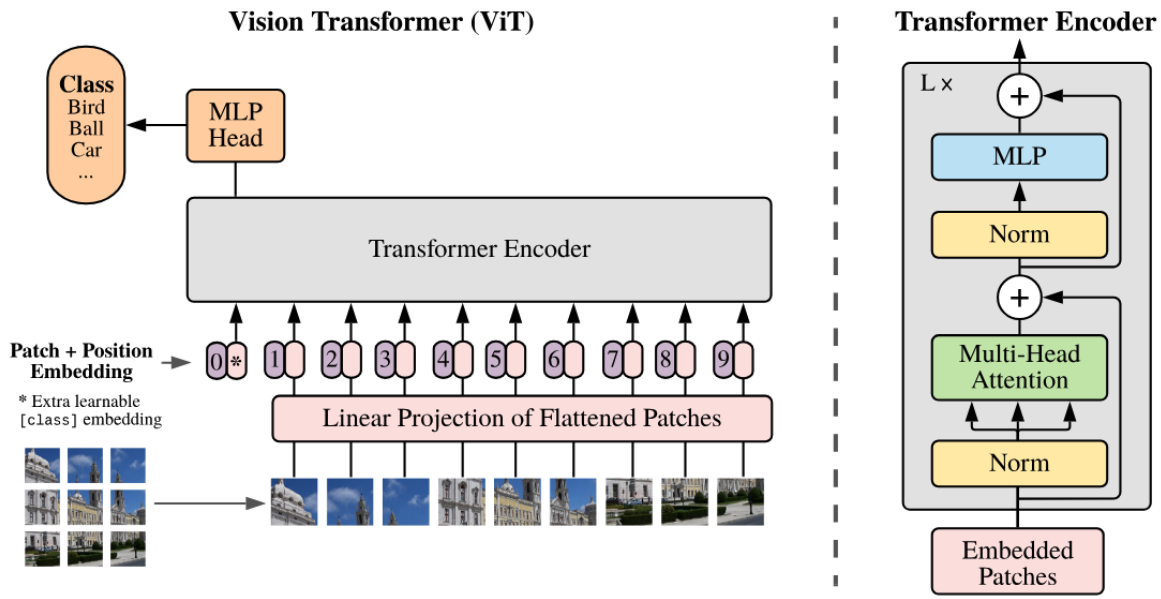
Applying Transformers to Images?

Images are arrays of pixels. Cut them in patches.



Use patches (p, p) as tokens.
For text, vocabulary was finite. Embedding layers were lookup tables of size $(|V|, d)$.
Instead use a linear map $R^{p \times p} \to \mathbb{R}^d$, i.e., an embedding matrix of size (p^2, d).

Visual Transformers (ViT)

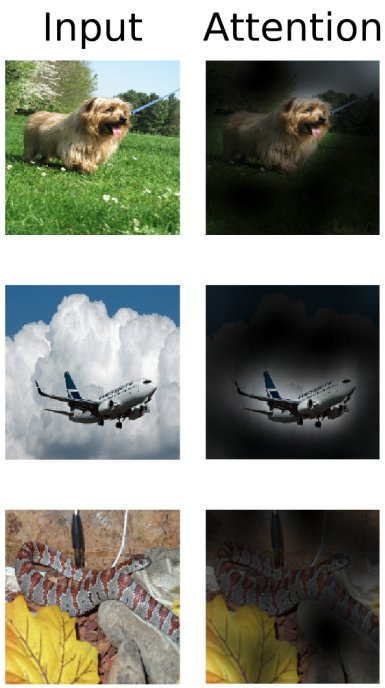


- **Idea:** treat an image as a sequence of patches.
- Split image into N patches (e.g. 16×16), **flatten** and **linearly embed.**
- Add positional encodings and pass through Transformer encoder.
- Optional `[CLS]` token for classification.

Difference with CNN

- Attention models spatial relationships directly — no convolutions.
- Global receptive field from the first layer.
- Need more data in general
- Complexity is quadratic in number of patches

Can visualize the attention on input of classification token:



CLIP: Connecting Vision and Language

- **CLIP (Contrastive Language–Image Pretraining)** — introduced by (Radford et al., 2021).
- Learns a **shared embedding space** for images and text by aligning them through contrastive learning.
- Trained on $\sim$400M (image, caption) pairs from the web.
- Objective: maximize similarity between matched image–text pairs, minimize for mismatched ones.

Training principle: Contrastive Learning

Given batch $\{(I_i, T_i)\}_{i=1}^{N}$:

$$\mathcal{L} = -\frac{1}{2N} \sum_i \left[\log \frac{e^{\text{sim}(f_I(I_i), f_T(T_i))/\tau}}{\sum_j e^{\text{sim}(f_I(I_i), f_T(T_j))/\tau}} + \log \frac{e^{\text{sim}(f_T(T_i), f_I(I_i))/\tau}}{\sum_j e^{\text{sim}(f_T(T_i), f_I(I_j))/\tau}} \right]$$

sim is cosine similarity, i.e, dot-product of the normalized vectors.

CLIP: Architecture and Applications

Architecture

- Two encoders:
  - Vision: ViT or ResNet : image embeddings $f_I(I)$
  - Text: Transformer : text embeddings $f_T(T)$
- Both encoders map to a single embedding vector ([EOS] or [CLS])
- Trained jointly but without task-specific labels.
- Alignment allows things like zero-shot transfer:
  - encode a picture
  - encode multiple captions: `'this is a cat','this is a dog'`
  - the embedding most similar to that of the picture is the best description.

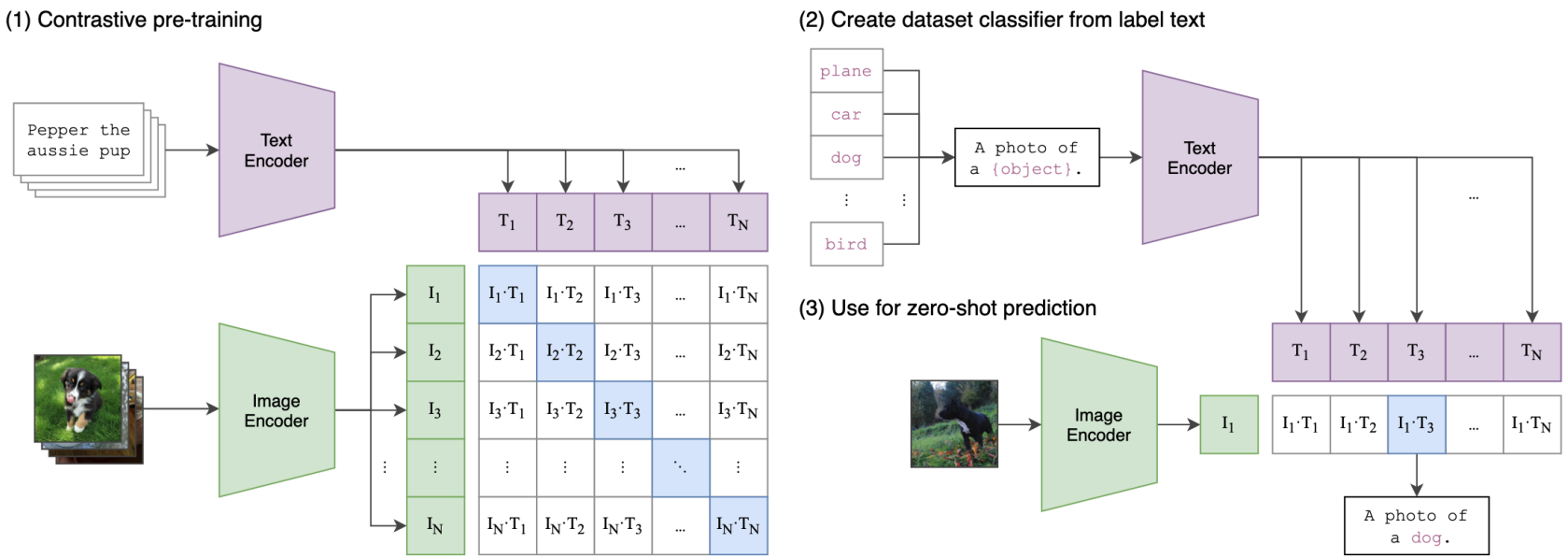


Table of Contents

Tokenization and Input Representation

- Transformers process sequences of **tokens**.
- Tokenization: convert data text into discrete units.
  - Character-level, word-level, or subword (BPE, WordPiece).
  - ``unbelievable'' → [``un'', ``believ'', ``able''].
- Embeddings: lookup table $E[t] \in \mathbb{R}^d$ for each token.
- Often same matrix used for input and output tokens.
- Handle variable lengths via **padding** and **masks**.

Example: Byte-Pair Encoding

- Subword algorithm that iteratively merges the most frequent adjacent symbol pairs to build a vocabulary.
- Keeps rare words decomposed into smaller units and common sequences as single tokens.

Algorithm:

1. Initialize vocabulary V with all characters (and a special end-of-word marker, e.g. `</w>`).
2. Represent the training corpus as sequences of characters terminated by `</w>`.
3. Repeat until $|V|$ reaches desired size:
   1. Count frequency of each adjacent symbol pair in the corpus.
   2. Select the most frequent pair (x, y).
   3. Add the merged symbol `xy` to V.
   4. Replace all occurrences of the pair xy by the merged symbol `xy` in the corpus representation.
4. Tokenization: segment new words by greedily applying the learned merges (longest-match first).