AI Applications Lecture 16

Image Generation AI 6: Text Encoders for Image Generation

SUZUKI, Atsushi
Jing WANG

Outline

Introduction

Roadmap Recap

Let's review what we have learned so far. The core of the image generation AI pipeline is the **reverse diffusion process** managed by the **denoising scheduler**. The reverse diffusion process receives information from the **text encoder** and passes the low-resolution latent image to the **natural image decoder**, which is composed of a VAE. Then, the natural image decoder (composed of a VAE) converts the low-resolution latent image into a natural image. This time, we will explain the **text encoder**.

Learning Outcomes

By the end of this lecture, students should be able to:

- **Control the output image** by **prompt weighting**, which processes the **output of the text encoder.**

Learning Outcomes

By the end of this lecture, students should be able to:

- **Control the output image** by **prompt weighting**, which processes the **output of the text encoder.**
- Explain the **objective function used to train the text encoders** employed in practical image generation AI.

Learning Outcomes

By the end of this lecture, students should be able to:

- **Control the output image** by **prompt weighting**, which processes the **output of the text encoder.**
- Explain the **objective function used to train the text encoders** employed in practical image generation AI.
- **Mathematically describe the architecture of the text encoder** used in practical image generation AI and explain its characteristics.

Preliminaries: Mathematical Notations

Preliminaries: Notations I

- **Definitions:**
  - $(\mathrm{LHS}) := (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a := b$ indicates that a is defined as b.

Preliminaries: Notations I

- **Definitions:**
  - $(\mathrm{LHS}) := (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a := b$ indicates that a is defined as b.
- **Set:**
  - Sets are often denoted by uppercase calligraphic letters. Example: $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that the element x belongs to the set $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (set extension notation).
  - $\{x \in \mathcal{A} \mid P(x)\}$: The set of elements in $\mathcal{A}$ for which the proposition $P(x)$ is true (set-builder notation).
  - $|\mathcal{A}|$: The number of elements in set $\mathcal{A}$ (used only for finite sets in this lecture).
  - $\mathbb{R}$: The set of all real numbers. $\mathbb{R}_{>0}$, $\mathbb{R}_{\geq 0}$ are defined similarly.
  - $\mathbb{Z}$: The set of all integers. $\mathbb{Z}_{>0}$, $\mathbb{Z}_{\geq 0}$ are defined similarly.
  - $[1, k]_{\mathbb{Z}}$: For $k \in \mathbb{Z}_{>0} \cup \{+\infty\}$, if $k < +\infty$, then $\{1, \ldots, k\}$; if $k = +\infty$, then $\mathbb{Z}_{>0}$.

Preliminaries: Notations II

- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$ denotes a mapping.
  - $y = f(x)$ denotes the output $y \in \mathcal{Y}$ for an input $x \in \mathcal{X}$.

Preliminaries: Notations II

- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$ denotes a mapping.
  - $y = f(x)$ denotes the output $y \in \mathcal{Y}$ for an input $x \in \mathcal{X}$.
- **Vector:**
  - Vectors are denoted by bold italic lowercase letters. Example: $\boldsymbol{v}$. $\boldsymbol{v} \in \mathbb{R}^n$.
  - The i-th component is written as v_i:

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}. \tag{1}$$

  - Standard inner product:

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle := \sum_{i=1}^{d_{\mathrm{emb}}} u_i v_i. \tag{2}$$

Preliminaries: Notations III

- **Sequence:**
  - $\boldsymbol{a} : [1, n]_{\mathbb{Z}} \to \mathcal{A}$ is called a sequence of length n. If $n < +\infty$, $\boldsymbol{a} = (a_1, \ldots, a_n)$; if $n = +\infty$, $\boldsymbol{a} = (a_1, a_2, \ldots)$.
  - The length is written as $|\boldsymbol{a}|$.

Preliminaries: Notations IV

- **Matrix:**
  - Matrices are denoted by bold italic uppercase letters. Example: $\boldsymbol{A} \in \mathbb{R}^{m,n}$.
  - Elements are denoted as $a_{i,j}$:

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}. \tag{3}$$

  - Transpose $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$:

$$\boldsymbol{A}^\top = \begin{bmatrix} a_{1,1} & \cdots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \cdots & a_{m,n} \end{bmatrix}. \tag{4}$$

  - Transpose of a vector:

$$\boldsymbol{v}^\top = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}. \tag{5}$$

Preliminaries: Notations V

- **Tensor:**
  - A tensor as a multi-dimensional array is denoted by an underlined bold italic uppercase letter $\underline{\boldsymbol{A}}$.

What Does the Text Encoder Receive, and What Information Does It Pass On?

Overview of the Pipeline

$$
\begin{tikzcd}
\left(\boldsymbol{t}^{[i]}\right)_{i=1}^{m} \arrow[r] & \mathsf{TextEnc}^{[i]}_{\boldsymbol{\alpha}^{[i]}} : \mathcal{V}^* \to \mathbb{R}^{d^{[i]}} \arrow[d, "\left(\boldsymbol{c}^{[i]}\right)_{i=1}^{m}"] & \\
\boldsymbol{u} \arrow[r] & \mathsf{LatentGen}_{\boldsymbol{\beta}} : \left(\prod_{i=1}^{m} \mathbb{R}^{d^{[i]}}\right) \times \mathcal{U} \to \mathcal{Z} \arrow[d, "\boldsymbol{z}"] & \\
& \mathsf{Dec}_{\boldsymbol{\gamma}} : \mathcal{Z} \to \mathcal{I} \arrow[r] & \underline{\boldsymbol{X}}
\end{tikzcd}
$$

Figure 1: Recap of the three-component pipeline.

First, let's clarify the input and output of a text encoder in general terms.

The input to the text encoder is originally a single byte sequence corresponding to the user's input prompt. Let's denote this as $\boldsymbol{b}$. In reality, a **tokenizer** (represented as the function $\mathsf{Tokenizer}$ below), often based on **Byte Pair Encoding (BPE)**, is used to convert each byte sequence into a token sequence $\boldsymbol{t}$. This is then fed into the neural network that constitutes the text encoder (hereafter $\mathsf{TextEncoderNN}_{\boldsymbol{\theta}_{\mathrm{TE}}}$). The output $C = \mathsf{TextEncoderNN}_{\boldsymbol{\theta}_{\mathrm{TE}}}(\boldsymbol{t})$ is a tensor that transforms the user's input prompt (or its corresponding token sequence) into a shape suitable for image generation.

Then it is passed to the **noise estimator** that constitutes the **reverse diffusion process.** Therefore, the size of the output tensor C must be acceptable as input by the noise estimator. If the noise estimator does not accept variable-sized inputs, the output size must be adjusted accordingly.

In practical applications, multiple prompts are input. For example, if there are a positive prompt $\boldsymbol{b}^+$ and a negative prompt $\boldsymbol{b}^-$, they are tokenized into $\boldsymbol{t}^+$ and $\boldsymbol{t}^-$ respectively, and both $\boldsymbol{C}^+ = \mathsf{TextEncoderNN}_{\boldsymbol{\theta}_{\mathrm{TE}}}(\boldsymbol{t}^+)$ and $\boldsymbol{C}^- = \mathsf{TextEncoderNN}_{\boldsymbol{\theta}_{\mathrm{TE}}}(\boldsymbol{t}^-)$ are passed to the noise estimator of the reverse diffusion process.

In the Case of Stable Diffusion 1.5

Example (Input/Output Dimensions in Stable Diffusion 1.5)

In text, let L_{tok} be the input token length and d_{ctx} be the embedding dimension. In the CLIP ViT-L/14 Text Encoder of Stable Diffusion 1.5, the **input length is fixed** at $L_{\text{tok}} = 77$, and the **embedding dimension is also fixed** at $d_{\text{ctx}} = 768$. Therefore,

$$\text{Input: } \boldsymbol{t} \in \{1, 2, ..., V\}^{77}, \qquad \text{Output: } \boldsymbol{C} \in \mathbb{R}^{77\times768}. \tag{6}$$

Here, $V = 49408$ is the vocabulary size. The specific values in the Hugging Face `CLIPTextModel` implementation are `Embedding(49408, 768)`, `position_embedding(77, 768)`, 12 Transformer layers, and the shape of the final output `last_hidden_state` is `(batch, 77, 768)`[1].

[1] The Diffusers/Transformers `CLIPTextModel` implementation conforms to `https://huggingface.co/openai/clip-vit-large-patch14` and returns `last_hidden_state` and `pooler_output`. See the CLIP documentation in Transformers at `https://huggingface.co/docs/transformers/en/model_doc/clip`.

Remark (Overview of Fixed-Length Conversion in SD1.5)

In practical Stable Diffusion (Diffusers), **padding/truncation to a fixed length $L = 77$ is performed during tokenization.** In the CLIP tokenizer settings, the **PAD (padding) token is set to be identical to the EOS** token. Furthermore, the **attention mask** is a binary sequence with 1 for non-PAD tokens and 0 for PAD tokens. Below, we rigorously define the fixed-length normalization function $\mathsf{LengthNormalization}^{(77)}$ and the attention mask generation function $\mathsf{MaskPAD}$ that masks PADs.

Length Normalization and PAD Mask

Definition (Length Normalization Function $\mathsf{LengthNormalization}^{(77)}$)

Let the **content tokens** (excluding special tokens) obtained from the input string be $\boldsymbol{s} = (s_1, \ldots, s_m)$, and abbreviate BOS (begin-of-sentence), EOS (end-of-sentence), and PAD (padding) as $\mathrm{BOS}, \mathrm{EOS}, \mathrm{PAD}$ respectively. Define the **preprocessed sequence** $\widetilde{\boldsymbol{t}}$ as $\widetilde{\boldsymbol{t}} := (\mathrm{BOS},\ s_1, \ldots, s_m,\ \mathrm{EOS})$ (addition of BOS/EOS). At this time, $\mathsf{LengthNormalization}^{(77)}$ is defined as follows:

$$\boldsymbol{t} = \begin{cases} (\widetilde{t}_1, \ldots, \widetilde{t}_\ell, \underbrace{\mathrm{PAD}, \ldots, \mathrm{PAD}}_{77-\ell \text{ items}}) & \text{if } \ell := |\widetilde{\boldsymbol{t}}| \leq 77, \\ (\widetilde{t}_1, \ldots, \widetilde{t}_{76},\ \mathrm{EOS}) & \text{if } |\widetilde{\boldsymbol{t}}| > 77. \end{cases} \tag{7}$$

Intuition of Length Normalization

That is, if the length is insufficient, it is right-padded with PAD (=EOS); if the length is excessive, the first 76 items are retained, and the last token is forcibly truncated to EOS.

PAD Mask Generation Function

Definition (PAD Mask Generation Function MaskPAD **(attention mask for PAD))**

For the normalized sequence $\boldsymbol{t} = (t_1, \ldots, t_{77})$ from equation (**??**), the function

$$\mathsf{MaskPAD} : \{1, \ldots, V\}^{77} \to \{0, 1\}^{77}, \tag{8}$$

$$\boldsymbol{a} = \mathsf{MaskPAD}(\boldsymbol{t}) \tag{9}$$

that generates the **attention mask** $\boldsymbol{a} \in \{0, 1\}^{77}$ is defined as

$$a_i := \begin{cases} 1 & \text{if } t_i \neq \mathrm{PAD}, \\ 0 & \text{if } t_i = \mathrm{PAD}, \end{cases} \qquad i \in [1, 77]_{\mathbb{Z}}, \tag{10}$$

$\boldsymbol{a}$ is used for **masking the key side** in the **self-attention** mechanism used downstream.

Controlling the Final Output Image Using Prompt Weighting

Basic Idea

The text encoder outputs a tensor, which is passed to the noise estimator that constitutes the reverse diffusion process. **Prompt weighting** is a technique that processes this tensor between the text encoder and the noise estimator, reflecting user intentions that cannot be fully expressed in the natural language input to the text encoder.

Prompt Weighting Syntax in Automatic1111

Example (Prompt weighting Syntax in Automatic1111)

Intuition for beginners: **Emphasizing** a part of the text makes the visual elements related to that word **more likely to appear strongly.** Conversely, de-emphasizing makes them **less likely to appear.** The typical syntax in Automatic1111 is as follows[2].

- Parentheses: `(word)` emphasizes by 1.1 times, multiplication is applied for nesting (e.g., `((word))` is $\approx 1.1^2$).
- Brackets: `[word]` de-emphasizes by 0.9 times (can be nested).
- Explicit weight: `(word:1.5)` specifies 1.5 times.

Expected effect: For example, in `(cinematic lighting:1.4), soft focus`, features related to `cinematic lighting` are supplied **more strongly** to the cross-attention, making the lighting expression more likely to be emphasized.

Diffusers and Compel

Remark

In **diffusers**, the above Prompt weighting syntax can also be used by employing the **Compel** library[3].

[3] Compel: `https://github.com/damian0815/compel`. Diffusers: `https://github.com/huggingface/diffusers`.

Text Encoder Class and I/O Correspondence

Definition (Text Encoder Class (Input/Output Size Correspondence))

A text encoder that takes a **token sequence** $\boldsymbol{t} \in \{1, 2, ..., V\}^{L_{\text{tok}}}$ as input and outputs a **context sequence** $\boldsymbol{C} \in \mathbb{R}^{L_{\text{ctx}} \times d_{\text{ctx}}}$ is defined as

$$\text{TextEncoder}_{\boldsymbol{\theta}_{\text{TE}}}^{(V,\ L_{\text{tok}},\ L_{\text{ctx}},\ d_{\text{ctx}})} :\ \{1, 2, ..., V\}^{L_{\text{tok}}} \to \mathbb{R}^{L_{\text{ctx}} \times d_{\text{ctx}}} \tag{11}$$

The CLIP ViT-L/14 of Stable Diffusion 1.5 has $L_{\text{tok}} = L_{\text{ctx}} = 77$, $d_{\text{ctx}} = 768$, $V = 49408$ (see Example 1).

Example (SD1.5 Text Encoder is Included in Definition 5)

In Definition 5, setting $(V, L_{\text{tok}}, L_{\text{ctx}}, d_{\text{ctx}}) = (49408, 77, 77, 768)$ and letting $\boldsymbol{\theta}_{\text{TE}}$ be all learnable parameters of CLIP ViT-L/14, the `last_hidden_state` of the `CLIPTextModel` in implementation provides $\boldsymbol{C} \in \mathbb{R}^{77 \times 768}$.

General Definition of Prompt Weighting

Definition (General Prompt Weighting (Action on Token Subsets))

Let the text encoder output be $\boldsymbol{C} = [\boldsymbol{c}_1^\top; \ldots; \boldsymbol{c}_{L_{\mathrm{ctx}}}^\top] \in \mathbb{R}^{L_{\mathrm{ctx}} \times d_{\mathrm{ctx}}}$. The index set $[1, L_{\mathrm{ctx}}]_{\mathbb{Z}}$ is partitioned into a family of **disjoint subsets** $\mathcal{S} = \{S_1, \ldots, S_K\}$ (i.e., $S_k \subseteq [1, L_{\mathrm{ctx}}]_{\mathbb{Z}}$, $S_i \cap S_j = \emptyset$, $\bigcup_{k=1}^K S_k = [1, L_{\mathrm{ctx}}]_{\mathbb{Z}}$). For each subset S_k, a mapping $\psi_k : \mathbb{R}^{d_{\mathrm{ctx}}} \to \mathbb{R}^{d_{\mathrm{ctx}}}$ is given (classes such as continuous maps or linear maps are assumed, but not required). At this time, the **weighting operation** $\mathrm{PW}_{\{\psi_k\},\mathcal{S}}$ is defined as

$$\widehat{\boldsymbol{C}} = \mathrm{PW}_{\{\psi_k\},\mathcal{S}}(\boldsymbol{C}) \in \mathbb{R}^{L_{\mathrm{ctx}} \times d_{\mathrm{ctx}}}, \tag{12}$$

$$\widehat{\boldsymbol{c}}_i = \psi_k(\boldsymbol{c}_i) \quad \text{if } i \in S_k \tag{13}$$

Equations (12)–(13) mean that for each token position i, the mapping ψ_k associated with the subset S_k it belongs to is applied to the component vector $\boldsymbol{c}_i$.

Example (Mapping ψ_k for Parentheses, Brackets, and Explicit Weights)

Suppose a parser extracts a family of subsets $\mathcal{S} = \{S_1, \ldots, S_K\}$ for text fragments (spans) and a scalar $r_k > 0$ for each S_k from the user prompt. At this time, if we define the **mapping** $\psi_k : \mathbb{R}^{d_{\text{ctx}}} \to \mathbb{R}^{d_{\text{ctx}}}$ as

$$\psi_k(\boldsymbol{x}) = r_k\, \boldsymbol{x} \tag{14}$$

then, according to Definition 7, a weight of r_k times is applied to the embedding $\boldsymbol{c}_i$ at the corresponding token position $i \in S_k$. The correspondence between Automatic1111-style syntax and r_k is as follows:

$$\texttt{(span)} \;\rightsquigarrow\; r_k = \lambda \quad (\lambda = 1.1 \text{ default; } \lambda^{\#\text{nest}} \text{ for nesting count}), \tag{15}$$

$$\texttt{(span: r)} \;\rightsquigarrow\; r_k = r \quad (r > 1). \tag{16}$$

Compel syntax

Example (Compel syntax)

In Compel, the syntax is as follows.

$$\texttt{span+} \rightsquigarrow r_k = \lambda \quad (\lambda = 1.1 \text{ default; } \lambda^{\#\text{nest}} \text{ for nesting count}), \tag{17}$$

$$\texttt{(span)r} \rightsquigarrow r_k = r \quad (r > 1). \tag{18}$$

In Compel, this r_k is expanded as **per-token weights** and applied to the embedding tensor in the form of equation (14)[4].

---

[4] One example of the application path: **per_token_weights** are constructed in `get_token_ids_and_expand_weights` of `embeddings_provider.py`, and reflected in the embeddings in `get_embeddings_for_weighted_prompt_fragments` (`https://github.com/damian0815/compel/blob/main/src/compel/embeddings_provider.py`).

Down-weighting is more complicated

Remark

In Compel's current implementation, **down-weighting** can be achieved with the following syntax:

$$\texttt{span-} \tag{19}$$

$$\texttt{(span)r} \quad (0 < \texttt{r} < 1). \tag{20}$$

Here, `(some phrase: 0.9)` is equivalent to `[some phrase]`. Interestingly, unlike up-weighting, the implementation of down-weighting is not composed of scalar multiplication of parts of the output, but is designed to use linear combinations with **another context output using attention masks**, and in this case, it does not stay within the scope of just processing the output (as it depends on things other than the original context output)[5].

[5] Changelog "1.0.0 - new downweighting algorithm": `https://pypi.org/project/compel/`.

Text Encoders in Practical Applications (Stable Diffusion 1.5's CLIP ViT-L/14 Text Encoder)

Terminology Note

Remark

CLIP (Contrastive Language-Image Pre-training) is a **learning method**, and **ViT-L/14** is an **architecture.** CLIP is a learning paradigm that encompasses multiple architectures such as ResNet/ViT [2, 1].

Overview via Diagram

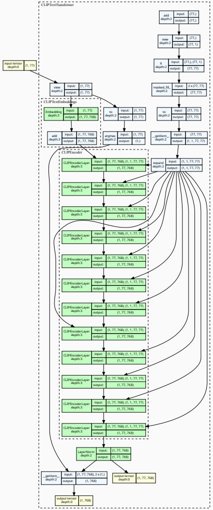

Figure 2: The neural network architecture of the CLIP ViT-L/14 text encoder in Stable

Rigorous Sub-layer Definitions i

Below, we use function names corresponding to library class names, denote **learnable parameters** in **parentheses**, and explicitly indicate **hyperparameters** in **superscript parentheses.** We will also state the sizes of all vectors, matrices, and tensors, including inputs, outputs, and intermediate variables.

Attention-related functions are defined to accept attention masks (The Transformers implementation applies `attention_mask` and `causal_attention_mask` sequentially)[6].

[6] In the Hugging Face Transformers CLIP implementation (`modeling_clip.py`), `causal_attention_mask` and `attention_mask` are combined and used in the text-side attention. Reference: `https://huggingface.co/transformers/v4.8.0/_modules/transformers/models/clip/modeling_clip.html`, latest documentation: `https://huggingface.co/docs/transformers/en/model_doc/clip`. For an application example using PyTorch's SDPA (scaled_dot_product_attention), see `https://huggingface.co/microsoft/LLM2CLIP-Openai-B-16/blob/main/modeling_clip.py`.

Embedding

Definition (Embedding**)**

Hyperparameters are vocabulary size $V \in \mathbb{Z}_{>0}$ and embedding dimension $d \in \mathbb{Z}_{>0}$. The learnable parameters are

$$\Theta_{\mathrm{Emb}} = \left(\boldsymbol{E} \in \mathbb{R}^{V \times d}\right). \tag{21}$$

Taking a token sequence $\boldsymbol{t} \in \{1, \ldots, V\}^L$ of length $L \in \mathbb{Z}_{>0}$ as input,

$$\text{Embedding}^{(V,d)}_{\Theta_{\mathrm{Emb}}} : \{1, \ldots, V\}^L \to \mathbb{R}^{L \times d}, \quad \text{Embedding}^{(V,d)}_{\Theta_{\mathrm{Emb}}}(\boldsymbol{t}) = \boldsymbol{X} \in \mathbb{R}^{L \times d}, \tag{22}$$

$$\text{where} \quad \boldsymbol{X}_{i,:} = \boldsymbol{E}_{t_i,:} \ \ (1 \le i \le L). \tag{23}$$

Layer Normalization

Definition (LayerNorm (Layer Normalization))

For an input $\boldsymbol{X} \in \mathbb{R}^{L \times d}$, the learnable parameters are

$$\Theta_{\mathrm{LN}} = \big(\boldsymbol{\gamma} \in \mathbb{R}^d,\ \boldsymbol{\beta} \in \mathbb{R}^d\big). \tag{24}$$

The mean and variance at each position $i \in \{1, \dots, L\}$ are

$$\mu_i = \frac{1}{d} \sum_{j=1}^{d} X_{i,j} \in \mathbb{R}, \quad \sigma_i^2 = \frac{1}{d} \sum_{j=1}^{d} \big(X_{i,j} - \mu_i\big)^2 \in \mathbb{R}, \tag{25}$$

and the output $\boldsymbol{Y} = \mathrm{LayerNorm}_{\Theta_{\mathrm{LN}}}(\boldsymbol{X}) \in \mathbb{R}^{L \times d}$ is defined as

$$Y_{i,j} = \gamma_j \frac{X_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} + \beta_j \quad (1 \leq i \leq L,\ 1 \leq j \leq d) \tag{26}$$

QuickGELU and Softmax

Definition (QuickGELUActivation (QuickGELU Activation))

For an element $u \in \mathbb{R}$,

$$\mathsf{QuickGELU}(u) = u\,\sigma(1.702\,u), \quad \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{27}$$

Definition ($\mathsf{Softmax}^{(i)}$ (Softmax; Axis Specification))

For a tensor (matrix) $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, specifying axis $i \in \{1, 2\}$,

$$\left(\mathsf{Softmax}^{(1)}(\boldsymbol{A})\right)_{p,q} = \frac{\exp(A_{p,q})}{\sum_{p'=1}^{m} \exp\left(A_{p',q}\right)} \in \mathbb{R}, \tag{28}$$

$$\left(\mathsf{Softmax}^{(2)}(\boldsymbol{A})\right)_{p,q} = \frac{\exp(A_{p,q})}{\sum_{q'=1}^{n} \exp\left(A_{p,q'}\right)} \in \mathbb{R} \tag{29}$$

CLIPAttention

Definition (CLIPAttention (Self-Attention; Mask-Compatible))

Takes input $\boldsymbol{X} \in \mathbb{R}^{L\times d}$ and **attention mask** $\boldsymbol{M} \in \{0,1\}^{L\times L}$. Hyperparameters are **number of heads** $h \in \mathbb{Z}_{>0}$ and **dimension per head** $d_h \in \mathbb{Z}_{>0}$ ($d = h\, d_h$). The learnable parameters are

$$\Theta_{\text{Attn}} = \left(\{\boldsymbol{W}_Q^{(\ell)} \in \mathbb{R}^{d\times d_h},\ \boldsymbol{W}_K^{(\ell)} \in \mathbb{R}^{d\times d_h},\ \boldsymbol{W}_V^{(\ell)} \in \mathbb{R}^{d\times d_h}\}_{\ell=1}^{h},\ \boldsymbol{W}_O \in \mathbb{R}^{(h\, d_h)\times d}\right). \tag{30}$$

For each head $\ell \in \{1,\ldots,h\}$,

$$\boldsymbol{Q}^{(\ell)} = \boldsymbol{X}\,\boldsymbol{W}_Q^{(\ell)} \in \mathbb{R}^{L\times d_h}, \quad \boldsymbol{K}^{(\ell)} = \boldsymbol{X}\,\boldsymbol{W}_K^{(\ell)} \in \mathbb{R}^{L\times d_h}, \quad \boldsymbol{V}^{(\ell)} = \boldsymbol{X}\,\boldsymbol{W}_V^{(\ell)} \in \mathbb{R}^{L\times d_h}, \tag{31}$$

$$\boldsymbol{S}^{(\ell)} = \frac{\boldsymbol{Q}^{(\ell)}(\boldsymbol{K}^{(\ell)})^\top}{\sqrt{d_h}} \in \mathbb{R}^{L\times L}. \tag{32}$$

Definition (CLIPAttention (continued))

Define **mask application** as

$$\big(\widetilde{\boldsymbol{S}}^{(\ell)}\big)_{i,j} = \begin{cases} S_{i,j}^{(\ell)} & \text{if } M_{i,j} = 1, \\ -\infty & \text{if } M_{i,j} = 0, \end{cases} \tag{33}$$

(where $-\infty$ implies the limit operation in softmax), and

$$\boldsymbol{A}^{(\ell)} = \mathrm{Softmax}^{(2)}\big(\widetilde{\boldsymbol{S}}^{(\ell)}\big)\,\boldsymbol{V}^{(\ell)} \in \mathbb{R}^{L\times d_h}, \tag{34}$$

$$\mathrm{CLIPAttention}_{\Theta_{\mathrm{Attn}}}^{(h,d_h)}(\boldsymbol{X},\boldsymbol{M}) = \big[\boldsymbol{A}^{(1)} \mid \cdots \mid \boldsymbol{A}^{(h)}\big]\boldsymbol{W}_O \in \mathbb{R}^{L\times d}. \tag{35}$$

(In the Transformers implementation, $\boldsymbol{M}$ is passed as a combination of `causal_attention_mask` and `attention_mask`, and applied internally with SDPA or an equivalent additive mask[7].)

[7] An example of mask composition in CLIP attention:

Definition (CLIPMLP (Per-Position MLP))

For input $\boldsymbol{X} \in \mathbb{R}^{L\times d}$, the intermediate dimension $d_{\mathrm{mlp}} \in \mathbb{Z}_{>0}$ is a hyperparameter, and the learnable parameters are

$$\Theta_{\mathrm{MLP}} = \left(\boldsymbol{W}_1 \in \mathbb{R}^{d\times d_{\mathrm{mlp}}},\ \boldsymbol{b}_1 \in \mathbb{R}^{d_{\mathrm{mlp}}},\ \boldsymbol{W}_2 \in \mathbb{R}^{d_{\mathrm{mlp}}\times d},\ \boldsymbol{b}_2 \in \mathbb{R}^{d}\right). \tag{36}$$

$$\boldsymbol{H}_1 = \boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}_L\boldsymbol{b}_1^\top \in \mathbb{R}^{L\times d_{\mathrm{mlp}}}, \tag{37}$$

$$\boldsymbol{H}_2 = \mathsf{QuickGELU}(\boldsymbol{H}_1) \in \mathbb{R}^{L\times d_{\mathrm{mlp}}}, \tag{38}$$

$$\mathsf{CLIPMLP}^{(d_{\mathrm{mlp}})}_{\Theta_{\mathrm{MLP}}}(\boldsymbol{X}) = \boldsymbol{H}_2\boldsymbol{W}_2 + \mathbf{1}_L\boldsymbol{b}_2^\top \in \mathbb{R}^{L\times d}. \tag{39}$$

CLIPEncoderLayer

Definition (CLIPEncoderLayer (Encoder Layer; Mask-Compatible))

The learnable parameters are

$$\Theta_{\mathrm{EL}} = \big(\Theta_{\mathrm{LN1}},\ \Theta_{\mathrm{Attn}},\ \Theta_{\mathrm{LN2}},\ \Theta_{\mathrm{MLP}}\big). \tag{40}$$

For input $\boldsymbol{X} \in \mathbb{R}^{L\times d}$ and attention mask $\boldsymbol{M} \in \{0,1\}^{L\times L}$,

$$\boldsymbol{U} = \mathsf{LayerNorm}_{\Theta_{\mathrm{LN1}}}(\boldsymbol{X}) \in \mathbb{R}^{L\times d}, \tag{41}$$

$$\boldsymbol{H} = \mathsf{CLIPAttention}^{(h,d_h)}_{\Theta_{\mathrm{Attn}}}(\boldsymbol{U},\boldsymbol{M}) \in \mathbb{R}^{L\times d}, \tag{42}$$

$$\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{H} \in \mathbb{R}^{L\times d}, \quad \boldsymbol{V} = \mathsf{LayerNorm}_{\Theta_{\mathrm{LN2}}}(\boldsymbol{Y}) \in \mathbb{R}^{L\times d}, \tag{43}$$

$$\boldsymbol{Z} = \mathsf{CLIPMLP}^{(d_{\mathrm{mlp}})}_{\Theta_{\mathrm{MLP}}}(\boldsymbol{V}) \in \mathbb{R}^{L\times d}, \quad \mathsf{CLIPEncoderLayer}_{\Theta_{\mathrm{EL}}}(\boldsymbol{X},\boldsymbol{M}) = \boldsymbol{Y} + \boldsymbol{Z} \in \mathbb{R}^{L\times d}. \tag{44}$$

(In Transformers, the corresponding layer `CLIPEncoderLayer` accepts

Definition (CLIPEncoder (Encoder Stack; Mask-Compatible))

The number of layers $N \in \mathbb{Z}_{>0}$ is a hyperparameter, and the learnable parameters are

$$\Theta_{\mathrm{Enc}} = \left(\Theta_{\mathrm{EL}}^{(1)}, \dots, \Theta_{\mathrm{EL}}^{(N)},\ \Theta_{\mathrm{LNf}}\right). \tag{45}$$

For input $\boldsymbol{X} \in \mathbb{R}^{L\times d}$ and attention mask $\boldsymbol{M} \in \{0,1\}^{L\times L}$,

$$\boldsymbol{H}_0 = \boldsymbol{X} \in \mathbb{R}^{L\times d}, \tag{46}$$

$$\boldsymbol{H}_\ell = \mathsf{CLIPEncoderLayer}_{\Theta_{\mathrm{EL}}^{(\ell)}}(\boldsymbol{H}_{\ell-1}, \boldsymbol{M}) \in \mathbb{R}^{L\times d} \quad (\ell = 1, \dots, N), \tag{47}$$

$$\mathsf{CLIPEncoder}_{\Theta_{\mathrm{Enc}}}^{(N)}(\boldsymbol{X}, \boldsymbol{M}) = \mathsf{LayerNorm}_{\Theta_{\mathrm{LNf}}}(\boldsymbol{H}_N) \in \mathbb{R}^{L\times d}. \tag{48}$$

(`CLIPTextTransformer` passes `attention_mask` to the `encoder`[9].)

[9] CLIP text transformer forward arguments:

CLIPTextEmbeddings

Definition ($\mathsf{CLIPTextEmbeddings}$ (Token + Position Embedding Sum))

Uses learnable parameters for token embedding $\Theta_{\mathrm{Tok}} = (\boldsymbol{E}_{\mathrm{tok}} \in \mathbb{R}^{V\times d})$ and learnable parameters for position embedding $\Theta_{\mathrm{Pos}} = (\boldsymbol{E}_{\mathrm{pos}} \in \mathbb{R}^{L\times d})$.

$$\boldsymbol{T} = \mathsf{Embedding}^{(V,d)}_{\Theta_{\mathrm{Tok}}}(\boldsymbol{t}) \in \mathbb{R}^{L\times d}, \tag{49}$$

$$\boldsymbol{P} = \mathsf{Embedding}^{(L,d)}_{\Theta_{\mathrm{Pos}}}\big([0, 1, \ldots, L-1]\big) \in \mathbb{R}^{L\times d}, \tag{50}$$

$$\mathsf{CLIPTextEmbeddings}^{(V,L,d)}_{\Theta_{\mathrm{Tok}},\Theta_{\mathrm{Pos}}}(\boldsymbol{t}) = \boldsymbol{T} + \boldsymbol{P} \in \mathbb{R}^{L\times d}. \tag{51}$$

The hyperparameters are

$$(V, L, d, h, d_h, N, d_{\mathrm{mlp}}) = (49408,\ 77,\ 768,\ 12,\ 64,\ 12,\ 3072) \quad \text{(where } d = h\, d_h\text{)} \tag{52}$$

The set of all learnable parameters is

$$\boldsymbol{\theta}_{\mathrm{TE}} = \left(\Theta_{\mathrm{Tok}},\ \Theta_{\mathrm{Pos}},\ \Theta_{\mathrm{Enc}}\right) \tag{53}$$

For input $\boldsymbol{t} \in \{1, \ldots, V\}^L$ and PAD mask sequence $\boldsymbol{a} = \mathsf{MaskPAD}(\boldsymbol{t})$ (Definition 3),

$$\boldsymbol{X}_0 = \mathsf{CLIPTextEmbeddings}^{(V,L,d)}_{\Theta_{\mathrm{Tok}},\Theta_{\mathrm{Pos}}}(\boldsymbol{t}) \in \mathbb{R}^{L\times d}, \tag{54}$$

$$\boldsymbol{C}_{\mathrm{causal}} = \left[\mathbf{1}_{\{j\le i\}}\right]_{i,j=1}^{L} \in \{0,1\}^{L\times L} \quad \text{(Lower-triangular causal r} \tag{55}$$

$$\boldsymbol{M}_{\mathrm{pad}} = \left[a_j\right]_{i,j=1}^{L} \in \{0,1\}^{L\times L} \quad \text{(Column replication of PAD m} \tag{56}$$

$$\boldsymbol{M} = \boldsymbol{C}_{\mathrm{causal}} \odot \boldsymbol{M}_{\mathrm{pad}} \in \{0,1\}^{L\times L}, \tag{57}$$

$$\boldsymbol{X}_N = \mathsf{CLIPEncoder}^{(N)}_{\Theta_{\mathrm{Enc}}}(\boldsymbol{X}_0, \boldsymbol{M}) \in \mathbb{R}^{L\times d}, \tag{58}$$

$$\mathsf{CLIPTextModel}^{(V,L,d,h,d_h,N,d_{\mathrm{mlp}})}_{\boldsymbol{\theta}_{\mathrm{TE}}}(\boldsymbol{t}) = \boldsymbol{X}_N \in \mathbb{R}^{L\times d}. \tag{59}$$

That is, M, which combines the **causal mask** and the **PAD mask (attention mask for padding)**, is passed to the self-attention in each layer. The Hugging Face implementation `CLIPTextModel` also accepts an `attention_mask` argument and applies it internally by combining it with the `causal_attention_mask`[10]. In Stable Diffusion, $C = X_N$ is supplied to the U-Net's **cross-attention**.

[10] CLIP text model API: `https://huggingface.co/docs/transformers/en/model_doc/clip`. Implementation example: `https://huggingface.co/transformers/v4.8.0/_modules/transformers/models/clip/modeling_clip.html`.

Mask Visualizations

Origin of Fixed Length

Remark

The fixed length $L = 77$ is stipulated by the position embedding $\mathtt{Embedding}^{(L,d)}$ (Equation (50)) and the tensor shaping based on it. This design is different from relative position embeddings or rotational position embeddings (RoPE), and this is the direct origin of the **fixed token length** in SD1.5 [2].

Training the CLIP Text Encoder

On Training Code Availability

Remark

Since the **actual training source code** used for the CLIP ViT-L/14 in Stable Diffusion 1.5 does not seem to be publicly available, we will explain based on `open_clip` (`mlfoundations/open_clip`) here[11].

[11] OpenCLIP repository: `https://github.com/mlfoundations/open_clip`.

CLIP Architecture and Objective Function i

Consider text sequences and image sequences of batch size N, $(\boldsymbol{t}_i,\ \underline{\boldsymbol{I}}_i)_{i=1}^N$.

- **Text Encoder** $\mathsf{TextEncoder}_{\boldsymbol{\theta}_{\mathrm{TE}}}:\ \{1,2,...,V\}^L \to \mathbb{R}^d$.
- **Image Encoder** $\mathsf{ImageEncoder}_{\boldsymbol{\theta}_{\mathrm{IE}}}:\ \mathcal{X}_{\mathrm{img}} \to \mathbb{R}^d$ (ViT/ResNet etc. [1]).
- **Temperature** $\tau > 0$ (often made learnable in implementations).

For each sample pair $(\boldsymbol{t}_i,\ \underline{\boldsymbol{I}}_i)$,

$$\boldsymbol{u}_i = \frac{\mathsf{TextEncoder}_{\boldsymbol{\theta}_{\mathrm{TE}}}(\boldsymbol{t}_i)}{\|\mathsf{TextEncoder}_{\boldsymbol{\theta}_{\mathrm{TE}}}(\boldsymbol{t}_i)\|_2}, \qquad \boldsymbol{v}_i = \frac{\mathsf{ImageEncoder}_{\boldsymbol{\theta}_{\mathrm{IE}}}(\underline{\boldsymbol{I}}_i)}{\|\mathsf{ImageEncoder}_{\boldsymbol{\theta}_{\mathrm{IE}}}(\underline{\boldsymbol{I}}_i)\|_2} \in \mathbb{R}^d. \tag{60}$$

CLIP Architecture and Objective Function ii

The similarity matrix $\boldsymbol{S} \in \mathbb{R}^{N\times N}$ is

$$S_{ij} = \frac{\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle}{\tau}. \tag{61}$$

The **cross-entropy losses** for text→image and image→text are

$$\mathcal{L}_{\text{Text}\to\text{Image}} = \frac{1}{N}\sum_{i=1}^{N}\Big(-\log\frac{\exp(S_{ii})}{\sum_{j=1}^{N}\exp(S_{ij})}\Big), \tag{62}$$

$$\mathcal{L}_{\text{Image}\to\text{Text}} = \frac{1}{N}\sum_{i=1}^{N}\Big(-\log\frac{\exp(S_{ii})}{\sum_{j=1}^{N}\exp(S_{ji})}\Big), \tag{63}$$

CLIP Architecture and Objective Function iii

and the **symmetric loss**

$$\mathcal{L}_{\text{CLIP}}(\boldsymbol{\theta}_{\text{TE}}, \boldsymbol{\theta}_{\text{IE}}, \tau) = \frac{1}{2}\left(\mathcal{L}_{\text{Text}\to\text{Image}} + \mathcal{L}_{\text{Image}\to\text{Text}}\right) \quad (64)$$

is **minimized.**

Similarity matrix

Similarity matrix $S = \left[\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle / \tau\right]_{i,j=1}^{N}$ (Ideal state)

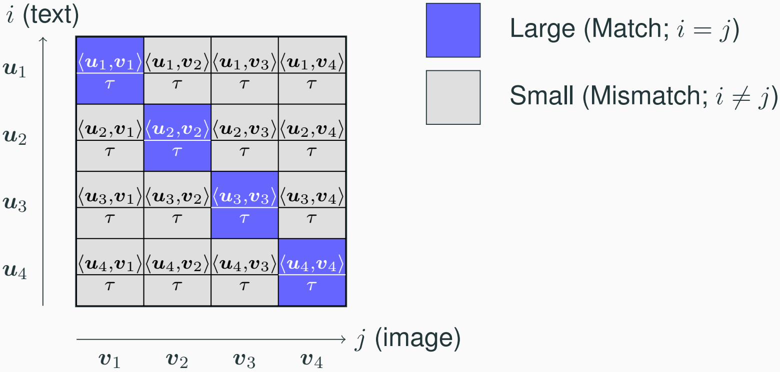

Figure 3: Visualization of the ideal similarity matrix.

Intuition of CLIP

Remark

Equation (64) applies pressure to make the inner product (cosine similarity) of the corresponding pair (i, i) **larger than others.** The temperature τ controls the sharpness of the softmax, and as $\tau \downarrow 0$, it approaches harder discrimination [2]. The OpenCLIP implementation conforms to this formulation[12].

[12] The OpenCLIP loss implementation is included in the training code at `https://github.com/mlfoundations/open_clip`.

Summary and Next Time

Correspondence with Learning Outcomes

- **Control via prompt weighting**: We formalized the weight action on token groups with Definition 7, and explained the correspondence with practical Automatic1111/Compel syntax in Examples 19–8 and Remark 3.

Correspondence with Learning Outcomes

- **Control via prompt weighting**: We formalized the weight action on token groups with Definition 7, and explained the correspondence with practical Automatic1111/Compel syntax in Examples 19–8 and Remark 3.
- **Explanation of objective function**: We specified CLIP's **symmetric contrastive learning loss** in Definition **??** and supplemented the intuition in Remark 7 (see [2]).

Correspondence with Learning Outcomes

- **Control via prompt weighting**: We formalized the weight action on token groups with Definition 7, and explained the correspondence with practical Automatic1111/Compel syntax in Examples 19–8 and Remark 3.
- **Explanation of objective function**: We specified CLIP's **symmetric contrastive learning loss** in Definition **??** and supplemented the intuition in Remark 7 (see [2]).
- **Mathematical description of architecture**: We **rigorously defined as functions** all layers of the **CLIP ViT-L/14 Text Encoder** in Definitions 10–**??**, and stated the origin of the fixed length in Remark 5 (see [2, 1]).

Next Time

From next time, we will describe the **learning (parameter update) of neural networks**.

References i

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby.
An image is worth 16x16 words: Transformers for image recognition at scale.
In International Conference on Learning Representations (ICLR), 2021.

[2] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever.
Learning transferable visual models from natural language supervision.

In Proceedings of the 38th International Conference on Machine Learning (ICML), 2021.
CLIP.