EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Discussion 12

1. Relative Positional Embedding

Year	Model Name	Type of Positional Embedding (PE)
2018, 2019, 2020	GPT-1, 2, and 3	Learned absolute PE
2021	Gopher	Relative PE (Transformer-XL)
2022	PaLM	RoPE
2023, 2024	LLaMA, LLaMA-2, and LLaMA-3	RoPE
2023	Grok-1	RoPE
2024	DeepSeek-V2, V3	decoupled RoPE
2025	LLaMA-4	iRoPE (interleaving RoPE and NoPE)
2025	Qwen-3	RoPE
2025	Gemma 3	RoPE

Table 1: Types of positional embeddings used in large language models over the years.

(a) **Why do we need positional encoding? Describe a situation where word order information is necessary for the task performed.**

Solution: Full, unmasked self-attention without positional cues is permutation equivariant: reordering the input vectors reorders the output vectors in the same way. Positional information lets the model distinguish word orders, which can change meaning. For example, "the man chased the dog" and "the dog chased the man" mean different things.

Absolute positional encodings assign a learned or fixed vector to each position. Relative methods instead describe relationships between positions. Causal masking can itself convey order information, as NoPE illustrates.

(b) **How does relative positional embedding work? How is it different from the absolute positional embedding?**

Solution: Relative positional methods make attention depend explicitly on the displacement between positions, rather than only their absolute indices. Examples include a learned attention bias indexed by $i - j$, learned relative-position vectors, and the relative phase introduced by RoPE. Absolute positional embeddings instead attach a vector to each individual position.

 All Rights Reserved. This may not be publicly shared without explicit permission.

(c) Consider a list of 2-dimensional vectors $\left[\left(x_1^{(1)}, x_1^{(2)}\right), \left(x_2^{(1)}, x_2^{(2)}\right), \left(x_3^{(1)}, x_3^{(2)}\right), \ldots\right]$. Now for any choice of parameter ω, the corresponding RoPE encoding is

$$\text{RoPE}\left(x_t^{(1)}, x_t^{(2)}, t\right) = \begin{pmatrix} \cos t\omega & -\sin t\omega \\ \sin t\omega & \cos t\omega \end{pmatrix} \begin{pmatrix} x_t^{(1)} \\ x_t^{(2)} \end{pmatrix} = \begin{pmatrix} x_t^{(1)} \cos t\omega - x_t^{(2)} \sin t\omega \\ x_t^{(2)} \cos t\omega + x_t^{(1)} \sin t\omega \end{pmatrix}$$

Equivalently, if we write the 2-dimensional vectors as complex numbers $z_t := x_t^{(1)} + jx_t^{(2)}$, then RoPE multiplies z_t by the unit complex phase $e^{jt\omega}$, where ω is a fixed real frequency:

$$\text{RoPE}\left(z_t, t\right) = e^{jt\omega} z_t$$

The positional dependence of the dot product is only on relative location:

$$\text{RoPE}(x, m)^T \, \text{RoPE}(y, n) = \text{RoPE}(x, m+k)^T \, \text{RoPE}(y, n+k)$$

for any integer k. This is obvious if we look at RoPE in the complex number form.

Extend RoPE to work for $2n$-dimensional vectors.

Solution: Given a $2n$-dimensional vector $(x^{(1)}, x^{(2)}, ..., x^{(2n)})$, represent it as an n-dimensional complex vector

$$(z^{(1)}, z^{(2)}, ..., z^{(n)}) := (x^{(1)} + jx^{(2)}, ..., x^{(2n-1)} + jx^{(2n)})$$

Then, we have

$$\text{RoPE}\left((z^{(1)}, z^{(2)}, ..., z^{(n)}), t\right) = \left(e^{jtw} z^{(1)}, e^{jtw} z^{(2)}, ..., e^{jtw} z^{(n)}\right)^T$$

or equivalently,

$$\text{RoPE}\left((x^{(1)}, x^{(2)}, ..., x^{(2n)}), t\right) = \begin{pmatrix} x^{(1)} \cos tw - x^{(2)} \sin tw \\ x^{(2)} \cos tw + x^{(1)} \sin tw \\ x^{(3)} \cos tw - x^{(4)} \sin tw \\ x^{(4)} \cos tw + x^{(3)} \sin tw \\ \vdots \\ x^{(2n-1)} \cos tw - x^{(2n)} \sin tw \\ x^{(2n)} \cos tw + x^{(2n-1)} \sin tw \end{pmatrix}$$

Each pair may choose a different w.

(d) **Does adding RoPE change the number of learnable parameters in the model? If so, how many new parameters are added (compared to a similar model with no positional encoding)?** **Solution:** Standard fixed-frequency RoPE adds no learnable parameters. The frequencies $\omega^{(i)}$ are fixed; a common choice is $\omega^{(i)} = 10000^{-2(i-1)/d}$ for a d-dimensional query or key vector.

Some models modify the frequency schedule or interleave RoPE and attention layers without explicit positional encodings. Such changes can improve long-context behavior, but generalization beyond the training context is not guaranteed by RoPE alone.

 All Rights Reserved. This may not be publicly shared without explicit permission.

2. Vision Transformer

A Vision Transformer (ViT) uses a Transformer encoder to classify images. It treats image patches as tokens, much as a text Transformer treats words or subwords as tokens. We consider the original supervised image-classification architecture:

1. **Split the image into patches.** Divide the image into non-overlapping $P \times P$ pixel patches. ViT/16 variants use $P = 16$.
2. **Form patch embeddings.** For an image with C channels ($C = 3$ for RGB), flatten each patch into P^2C numbers. Apply the same learned linear projection to every patch to obtain a D-dimensional token embedding.
3. **Add a class token and positional information.** Prepend a learned classification (CLS) token and add learned absolute positional embeddings to the patch tokens and the CLS token.
4. **Encode and classify.** Pass the sequence through Transformer encoder blocks containing multi-head self-attention and feed-forward networks. Feed the final CLS representation to a classification head to predict the image's label.

The encoder uses full, unmasked self-attention. The questions below explore this classifier and how its architecture or training objective could be adapted to other tasks.

Below is a diagram of ViT for supervised image classification.

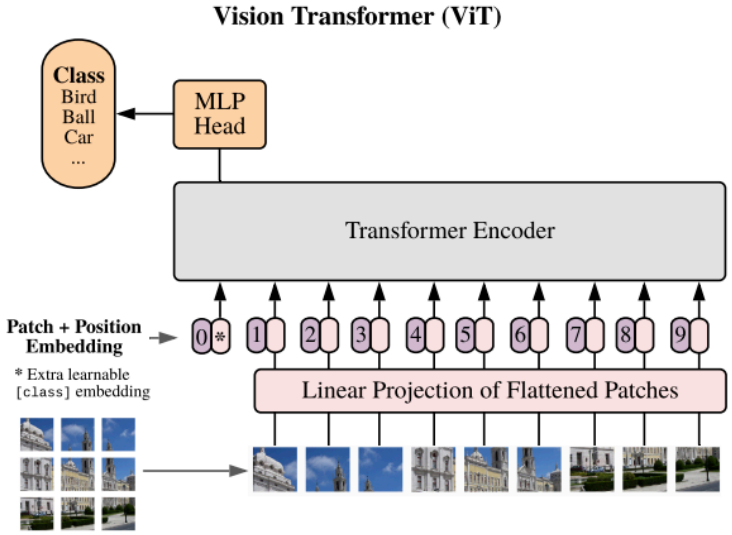


Figure 1: Vision Transformer (Link to Google blog)

(a) Does it matter which order you flatten the sequence of patches?

Solution: No, the positional encoding will tell the model where the patch came from, as long as we use the same flattening order (ie. row-order or column-order) throughout training and inference.

(b) What is the complexity of the vision transformer attention operation? Assume you have an image of size $H \times W$ and patches of size $P \times P$. Only consider the time of the attention operation, not the time to produce queries, keys, and values. Queries, keys, and values each have dimension D.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Solution: We have $(H/P) \times (W/P)$ patches, so the complexity is $O(\text{seq}^2 D) = O((HW/P^2)^2 D)$.

(c) What is the receptive field of one sequence item after the first layer of the transformer? How does this compare to a CNN, and what are the pros and cons of this?

Solution: After a single transformer layer, the receptive field is the entire image. In contrast, a CNN's receptive field grows slowly through the layers. This change allows transformers to pick up on spatially distant patterns more easily. However, the CNN's inductive bias toward spatially local patterns can be helpful for learning, especially when training from scratch on a small dataset. Some transformers (like Swin Transformer) have also explored only attending to spatially-local patches in early layers of the transformer to recover this inductive bias.

(d) If we forgot to include positional encodings, could the model learn anything at all? State one task where a model could perform well without positional encoding, and one task where it would do poorly.
Solution: The model would see the image as a bag of patches (similar to a bag of words in NLP). It could still perform well on tasks where patches are sufficient to understand the content of the image (e.g. classifying traffic lights as red, yellow, or green), but it would perform poorly on tasks that rely on the order of patches (e.g. reading text from a screen).

(e) If you wanted to add a few conv layers into this architecture, how would you incorporate them?

Solution: There's no one unique solution to this. One way is to first apply a few conv layers to the input image, and then split the resulting feature map into patches for the rest of the ViT operations[1]. Research found that applying convolution for the early "patchify" layer leads to more robust ViT training. Another option is to add depthwise convolutions inside the feed-forward blocks to introduce locality, as in SegFormer; overlapping patch embeddings between stages can reduce spatial resolution[2].

(f) How would you use this architecture to do GPT-style autoregressive generation of images?

Solution: Use causal attention masking and an output head that predicts the next image patch from previous patches. Train with shifted next-patch targets, using MSE against the next ground-truth patch for continuous predictions, or cross-entropy for discrete image tokens (e.g. Parti, `https://sites.research.google/parti/`). At inference time, feed each generated patch or token back into the sequence to generate the next one.

(g) So far the vision transformer training procedure has been fully-supervised. Yet we know that most of the available data are unlabelled. How can we do BERT-style self-supervised **representation** learning with vision transformers?

- *Hint 1*: Think about how BERT is trained. How should the input image be modified? What should be the target?
- *Hint 2*: ViT in this question only has an encoder. For masked-patch training, add a patch-prediction head or decoder. What must it predict?

Solution: This question also does not have a unique solution. Instead, the following solution provides an overview of proven effective methods in past research.

BERT is able to obtain useful representations for NLP tasks using two pretext tasks: (1) mask prediction (2) next sentence prediction, with mask prediction providing the direct analogy here. In vision transformer, the analog of mask prediction will be **predicting the masked patches**.

Specifically, we can mask out, say, 50% of the input patches and treat the masked image as input to our model.

[1] Xiao, Tete, et al. "Early convolutions help transformers see better." Advances in neural information processing systems 34 (2021): 30392-30400.

[2] Xie, Enze, et al. "SegFormer: Simple and efficient design for semantic segmentation with transformers." Advances in Neural Information Processing Systems 34 (2021): 12077-12090.

 All Rights Reserved. This may not be publicly shared without explicit permission.

The model will be trained to predict the masked region of the original image, by optimizing a reconstruction loss (e.g. mean squared error).

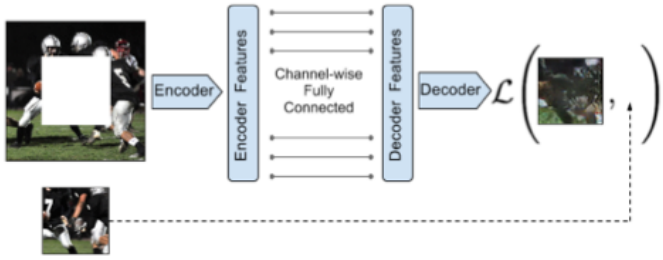


Figure 2: Diagram of Context Encoder

This objective of predicting masked patches or pixels has a technical name called **inpainting**, and the aforementioned procedure is a simplified version of **Context Encoder** (Pathak et al), which takes in a masked image (with masked pixels filled with a constant mean value) and predicts the missing region. MAE uses a different design: **mask tokens are not fed into MAE's image encoder**.

One motivation for keeping mask tokens out of MAE's image encoder is that at test-time, real images do not contain such a huge portion of mask values. This creates a distribution difference between masked training inputs and unmasked downstream inputs. MAE instead only feeds the encoder *real* patches and then appends the mask tokens / values right before the decoder input for image reconstruction.

The original approach uses **image inpainting**. Specifically, we set the pixels that we wish to mask to a constant value (e.g., zero). A downside of this approach is that it requires spending compute on masked out regions, which can be large and contain little useful information. MAE uses an *asymmetric* encoder-decoder model and only feeds the *visible* patches to the encoder and includes mask tokens, restored to their original positions with positional embeddings, as decoder inputs for image reconstruction.

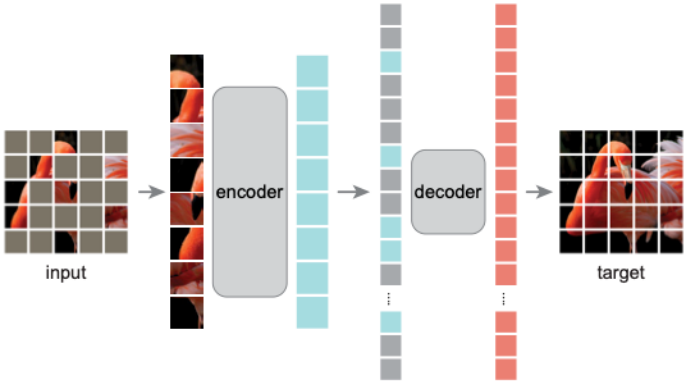


Figure 3: Diagram of Masked Autoencoder

This is the idea of Masked Autoencoder or MAE[3]. MAE uses a Vision Transformer encoder-decoder structure: it splits an image into patches, randomly masks out 75% of them, passes the visible patches through the encoder, and restores mask tokens at their original positions before the decoder. The reconstruction loss is computed on the masked patches.

Contributors: Matteo Guarrera, Mert Cemri, Sizhe Chen, Gireeja Ranade, Kevin Li, Anant Sahai, Olivia Watkins, Jerome Quenum, Saagar Sanghavi, Yuxi Liu, Qiyang Li.

[3]He, Kaiming, et al. "Masked autoencoders are scalable vision learners." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

 All Rights Reserved. This may not be publicly shared without explicit permission.