# Deep Learning (1470)

Randall Balestriero

Class 13: Transformers

# Recap!


Recurrent Neural Network with Pooling Architecture
y_t
Pooled representation
Pool (last K steps)
K=3
h_{t-3}
h_{t-2}
h_{t-1}
h_t
h_{t+1}
x_{t-3}
x_{t-2}
x_{t-1}
x_t
x_{t+1}
t-3
t-2
t-1
t
t+1
Input (x_t)
Hidden State (h_t)
Output (y_t)
Pooling Mechanism

# Transformers anyone?

# Transformers anyone?


SINGLE HEAD SELF-ATTENTION MECHANISM
INPUT
THREE PARALLEL PROJECTIONS
ATTENTION COMPUTATION
OUTPUT
$x_1$
$x_2$
$x_3$
...
$x_T$
X
$(T \times d)$
T tokens, each d-dimensional. Token Embeddings.
$W_Q$
$(d \times d_k)$
$W_K$
$(d \times d_k)$
$W_V$
$(d \times d_k)$
Q
$(T \times d_k)$
Queries (Q)
K
$(T \times d_k)$
Keys (K)
V
$(T \times d_k)$
Values (V)
softmax $\left(\frac{QK^T}{\sqrt{d_k}}\right)$
$(T \times T)$
High Attention
Low Attention
A (Attention Matrix)
Attention Weights
$(\mathbf{A} = \mathrm{softmax}(\mathbf{QK}^T/\sqrt{d_k}))$
$\mathbf{Z} = \mathbf{A} \cdot \mathbf{V}$
Z
$(T \times d_k)$
Output (Z)
$z_1$
$z_2$
$z_3$
...
$z_T$
Contextualized Representations.
CONCRETE EXAMPLES OF ATTENTION PATTERNS FOR LANGUAGE MODELING
EXAMPLE 1 (t=3): Short Sentence
$x_1$ The $x_2$ cat $x_3$ sat
Strong Attention
t=1 The cat sat
Attention 0.1 0.7 0.2
Position t=3 ("sat") attends strongly to "cat" (the subject).
EXAMPLE 2 (t=15): Long-range Dependency
t=1 t=3 t=5 t=15 t=13
The curious cat that lived in the old house on the hill sat quietly
Long-range
t=1 The cat lived house on hill quietly
Attention 0 0 7 0 0 0 0 0 0 0 0 0 0 6 0
Sparse attention; Position t=15 ("sat") attends to "cat" despite distance.
EXAMPLE 3 (Question): Question Answering
$x_1$ What $x_2$ color $x_3$ is $x_4$ the $x_5$ sky
Attends to Question Word
t=1 What color is the sky
Attention
Position t=5 ("sky") attends back to "What" and "color" to form the answer.
COLOR LEGEND: Light Blue = Queries (Q), Light Grein = Keys (K) = Values (V) = Output (Z) & Attention Weights = Low Attention = High Attention

Multi-Head Self-Attention
Input Sequence
$x_1$
$x_2$
$\vdots$
$x_T$
Input Sequence
$\mathbf{X}$
$\mathbf{X}$
$(T \times d)$
Head 1
$\mathbf{W}_Q^{(1)}$ $(d \times d_k)$
$\mathbf{W}_K^{(1)}$ $(d \times d_k)$
$\mathbf{W}_V^{(1)}$ $(d \times d_k)$
$\mathrm{softmax}(\mathbf{QK}^T/\sqrt{d_k})\cdot\mathbf{V}$
Head 2
$\mathbf{W}_Q^{(2)}$ $(d \times d_k)$
$\mathbf{W}_K^{(2)}$ $(d \times d_k)$
$\mathbf{W}_V^{(2)}$ $(d \times d_k)$
$\mathrm{softmax}(\mathbf{QK}^T/\sqrt{d_k})\cdot\mathbf{V}$
Head 3
$\mathbf{W}_Q^{(3)}$ $(d \times d_k)$
$\mathbf{W}_K^{(3)}$ $(d \times d_k)$
$\mathbf{W}_V^{(3)}$ $(d \times d_k)$
$\mathrm{softmax}(\mathbf{QK}^T/\sqrt{d_k})\cdot\mathbf{V}$
Head 4
$\mathbf{W}_Q^{(4)}$ $(d \times d_k)$
$\mathbf{W}_K^{(4)}$ $(d \times d_k)$
$\mathbf{W}_V^{(4)}$ $(d \times d_k)$
$\mathrm{softmax}(\mathbf{QK}^T/\sqrt{d_k})\cdot\mathbf{V}$
$\mathbf{Z}^{(1)}$ Head Output $(T \times d_k)$
$\mathbf{Z}^{(2)}$ Head Output $(T \times d_k)$
$\mathbf{Z}^{(3)}$ Head Output $(T \times d_k)$
$\mathbf{Z}^{(4)}$ Head Output $(T \times d_k)$
Concat
$(T \times H\cdot d_k)$
$= (T \times 4d_k)$
$\mathbf{W}_O$
$(H\cdot d_k \times d)$
$\mathbf{Z}$
$\mathbf{Z}$
$(T \times d)$
Multi-Head Output
$(T \times d)$
$z_1$
$z_2$
$\vdots$
$z_T$
- $d_k = d$ / H
- Example: d=512, H=8 → $d_k$=64
- Each head: different attention pattern
$\mathrm{MultiHead}(\mathbf{X}) = \mathrm{Concat}(\mathrm{head}_1, \mathrm{head}_2, \ldots, \mathrm{head}_H)\cdot\mathbf{W}_O$ where $\mathrm{head}_h = \mathrm{Attention}(\mathbf{XW}_Q^{(h)}, \mathbf{XW}_K^{(h)}, \mathbf{XW}_V^{(h)})$
Why Multiple Heads?
Head 1: Syntactic
Head 2: Semantic
Head 3: Local
Head 4: Long-range
T×T
T×T
1
0
Light Blue: Query (Q)
Light Green: Key (K)
Light Orange: Value (V)
Light Purple: Attention
Gray Text: Shapes

# Break: Remember batch-norm?

# Batch Normalization vs Layer Normalization

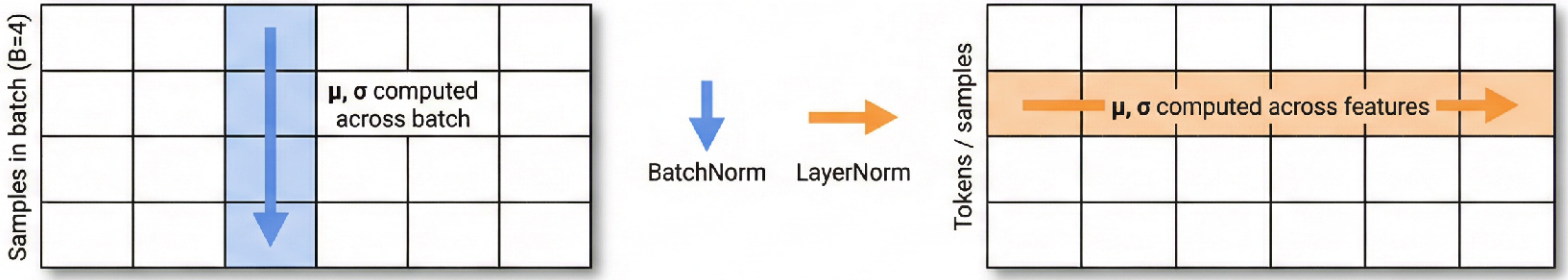
## Why transformers use LayerNorm

BATCH NORMALIZATION

(Batch/Tokens × Features)

LAYER NORMALIZATION

(Batch/Tokens × Features)




### Batch Normalization

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu_B}{\sigma_B}$$

Per feature, across batch

### Layer Normalization

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu_d}{\sigma_d}$$

Per token, across features

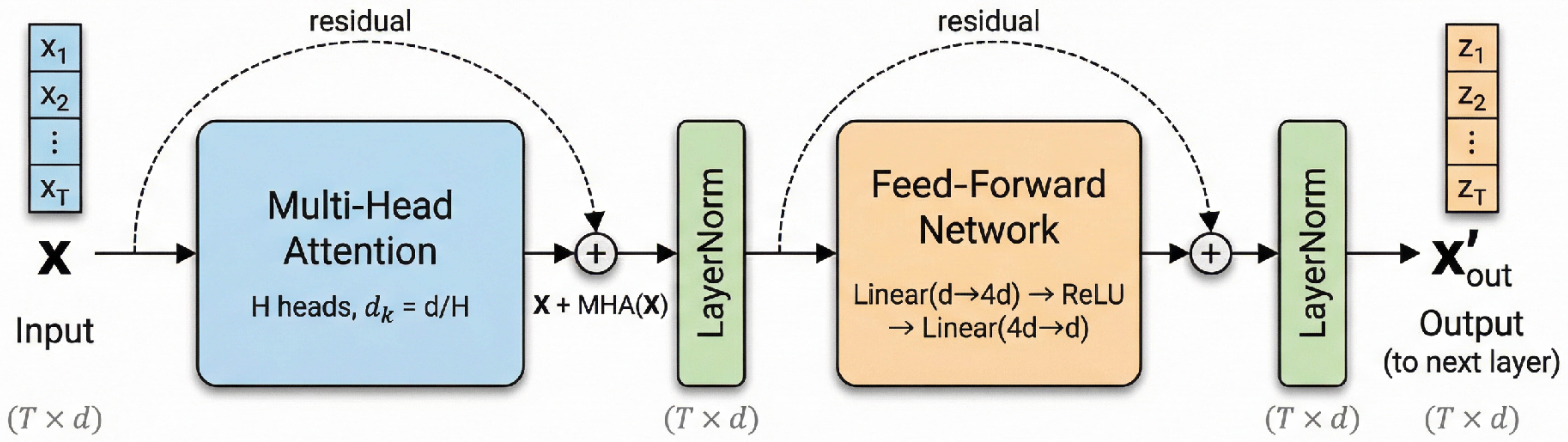
| | **Batch Norm** | **Layer Norm** |
|---|---|---|
| Normalizes | Across batch (↓) | Across features (→) |
| Statistics | Per feature | Per token |
| Batch size 1 | ❌ Fails | ✓ Works |
| Train=Inference | ❌ Different | ✓ Same |
| Transformers | ❌ Not used | ✓ Standard |

**LayerNorm**: each token normalized independently → perfect for variable-length sequences & autoregressive generation

$$\text{LayerNorm}(\mathbf{x}) = \gamma \cdot \frac{(\mathbf{x} - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

# Transformer Layer

Stacked N× times




Self-Attention Sublayer:

$$\mathbf{X}' = \mathrm{LayerNorm}(\mathbf{X} + \mathrm{MultiHeadAttention}(\mathbf{X}))$$

Feed-Forward Sublayer:

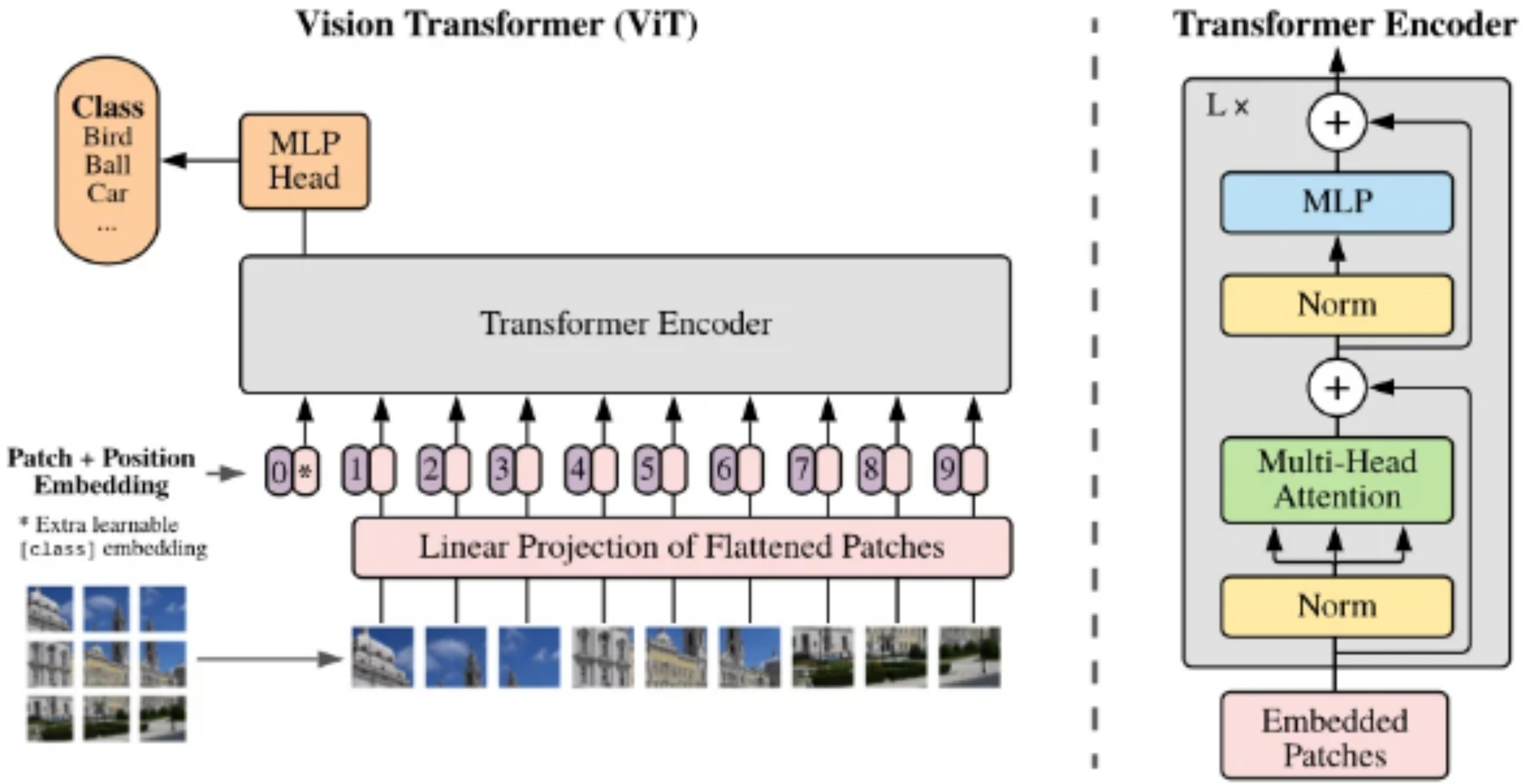
$$\mathbf{X}'' = \mathrm{LayerNorm}(\mathbf{X}' + \mathrm{FFN}(\mathbf{X}'))$$

$$\mathrm{FFN}(\mathbf{x}) = \mathrm{W}_2 \cdot \mathrm{ReLU}(\mathrm{W}_1\mathbf{x} + \mathrm{b}_1) + \mathrm{b}_2$$

d = model dim (512, 768)
$d_{ff}$ = 4d (2048, 3072)
H = heads (8, 12)

Light Blue: Attention Components
Light Green: LayerNorm
Light Orange: Feed-Forward Network

# Vision Transformers!



# Break: efficient implementation for sequence modeling?

# Causal Self-Attention

Masked attention for autoregressive models (GPT, LLaMA, etc.)

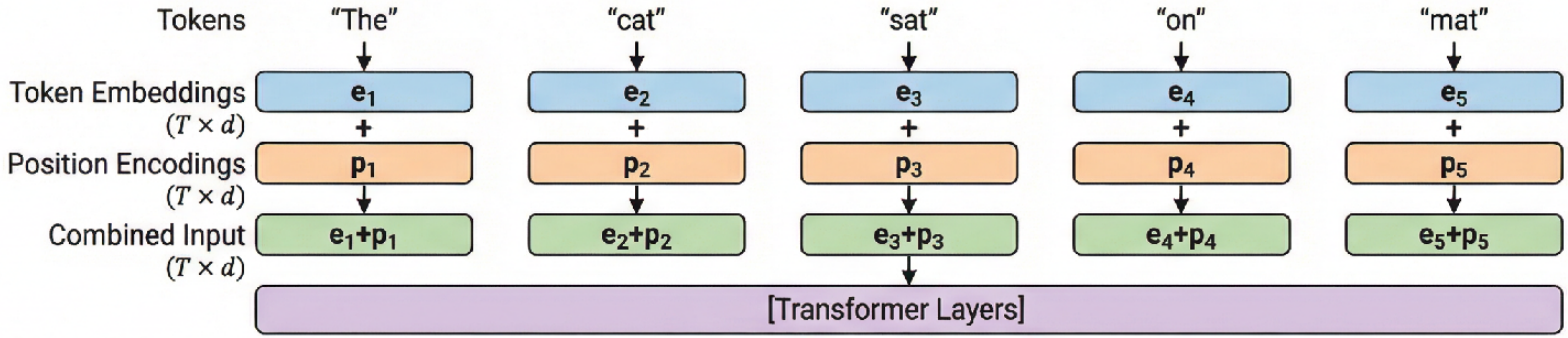

Causal Attention: $\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{Q}\cdot\mathbf{K}^T + \mathbf{M}}{\sqrt{d_k}}\right)$ $\quad \mathbf{Z} = \mathbf{A}\cdot\mathbf{V}$ $\quad$ where $M_{ij} = 0$ if $j \le i$, else $-\infty$

# Break: How to encode position?

# Positional Encodings

Injecting position information into the transformer

## SECTION 1: WHERE TO ADD



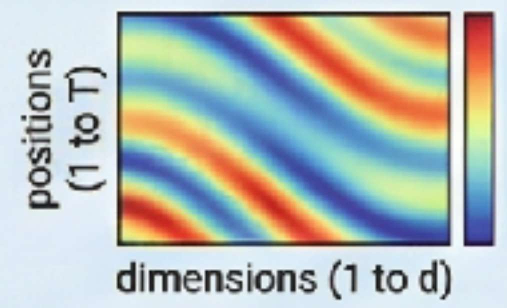

Position encoding added to token embeddings **BEFORE** transformer layers

## SECTION 2: TYPES OF POSITIONAL ENCODINGS

### **Sinusoidal** (Original Transformer)

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

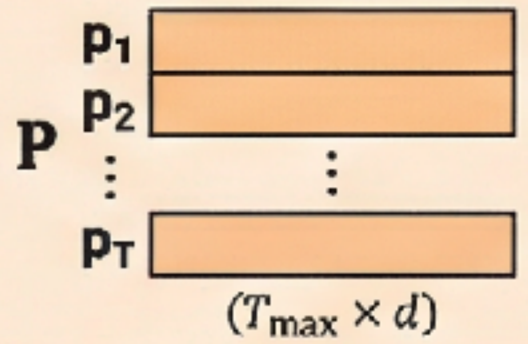



- Fixed (not learned)
- Deterministic
- Can extrapolate to longer sequences

### **Learned** (BERT, GPT-2)

$$\mathbf{P} \in \mathbb{R}^{T_{\max} \times d}$$

"Lookup table of learnable vectors"

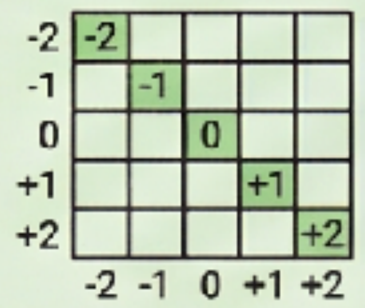



- Learned during training
- More flexible
- Limited to max sequence length $T_{max}$

### **Relative** (Transformer-XL, T5)

"Encode relative distance $(i - j)$ not absolute position"

$a_{ij}$ depends on $(i - j)$



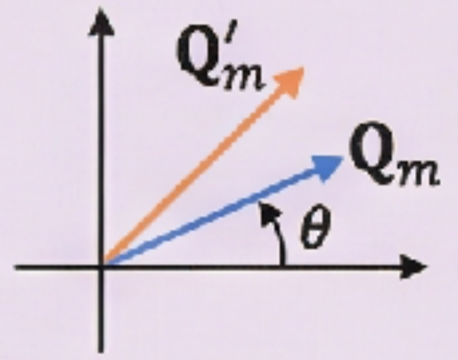

- Captures relative distance
- Better for long sequences
- Added in attention computation

### **RoPE / Rotary** (LLaMA, GPT-NeoX)

"Rotate **Q** and **K** vectors based on position"

$$\mathbf{Q}'_m = \mathbf{R}_m \cdot \mathbf{Q}_m$$

$$\mathbf{K}'_n = \mathbf{R}_n \cdot \mathbf{K}_n$$




- Applied to **Q, K** in attention
- Relative position via rotation
- Extrapolates well

## SECTION 3: VISUAL COMPARISON

| Method | Where Added | Learned? | Extrapolation |
|---|---|---|---|
| Sinusoidal | Input | No | ✓ Good |
| Learned | Input | Yes | ✗ Limited |
| Relative | Attention | Yes | ✓ Good |
| RoPE | Q, K | No | ✓ Good |

## EQUATIONS BOX

**Input to transformer:**

$$\mathbf{X} = \text{TokenEmbed(tokens)} + \text{PositionEncode(positions)}$$

**For sinusoidal:**

$$PE(t, 2i) = \sin\left(\frac{t}{10000^{2i/d}}\right) \qquad PE(t, 2i+1) = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

See you Monday!