NUS | Computing
National University of Singapore

IT5005 Artificial Intelligence

Sirigina Rajendra Prasad
AY2025/2026: Semester 1

Tutorial 9: Transformers

Data Preparation: Corpus and Dictionary

- Dictionary class
  - Bidirectional mapping from token ID to token
- Corpus class
  - Converts given text data into a sequence of tokens

Data Preparation

- Data is loaded in batches

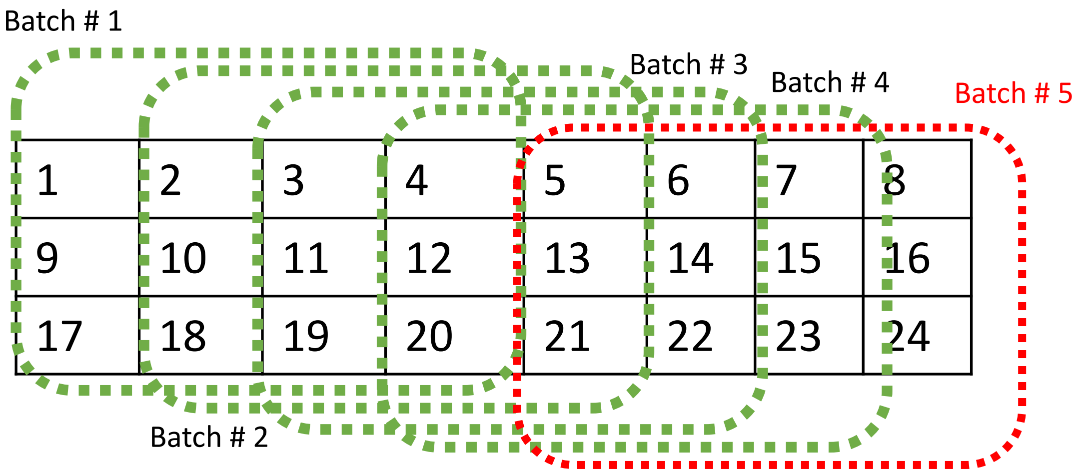
- Two paramaters to define a batch: $batch\_size$ and $sequence\_length$
  - $batch\_size$ is the number of parallel sequences in a batch that are processed simultaneously
  - $sequence\_length$ is the length of the sequence in a batch

- Each batch is of dimension $(batch\_size, sequence\_length)$

Data Preparation

- $data\_size$ = Total available tokens
- $batch\_size \quad =$ Number of parallel sequences
- $tokens\_per\_sequence \ = \dfrac{data\_size}{batch\_size}$ -> integer division. (_prepare_data method)
- seq_len = window size
- $num\_batches \ = \ tokens\_per\_sequence - \ seq\_length$
- $start\_idx$ = starting position of the window
- Constraints
  - $seq\_len \leq tokens\_per\_sequence$
  - $start\_idx \leq tokens\_per\_sequence \ - \mathrm{seq\_len} \ - \ 1$

Data Preparation



- Example:
  - $\text{corpus.data} = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26]$
  - $seq\_len \ = \ 4 \quad (\text{window size} = 4 \text{ for sliding window})$
  - $num\_batches = tokens\_per\_sequence - seq\_length = 8 - 4 = 4$
- We cannot use Batch #5. Why?
  - No target for Token 24.

Preparing Data for Training

[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26]

$$tokens\_per\_sequence = \frac{data\_size}{batch\_size} = 26//3 = 8$$

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24

Batchification

Input Data

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24

Batch #0

Input ($batch\_size = 3$)

1	2	3	4
9	10	11	12
17	18	19	20

$seq\_len = 4$

Target ($batch\_size = 3$)

2	3	4	5
10	11	12	13
18	19	20	21

$seq\_len = 4$

Batch #1

2	3	4	5
10	11	12	13
18	19	20	21

$seq\_len = 4$

3	4	5	6
11	12	13	14
19	20	21	22

$seq\_len = 4$

Batch #2

3	4	5	6
11	12	13	14
19	20	21	22

$seq\_len = 4$

4	5	6	7
12	13	14	15
20	21	22	23

$seq\_len = 4$

Batch #3

4	5	6	7
12	13	14	15
20	21	22	23

$seq\_len = 4$

5	6	7	8
13	14	15	16
21	22	23	24

$seq\_len = 4$

$get\_batch(i)$ would pick i −th batch

Batch #0

Input

$batch\_size = 3$

1	2	3	4
9	10	11	12
17	18	19	20

$seq\_len = 4$

Target

$batch\_size = 3$

2	3	4	5
10	11	12	13
18	19	20	21

$seq\_len = 4$

Input	Target
1	2
1,2	3
1,2,3	4
1,2,3,4	5
9	10
9,10	11
9,10,11	12
9,10,11,12	13
17	18
17,18	19
17,18,19	20
17,18,19,20	21

Batch #1

Input

2	3	4	5
10	11	12	13
18	19	20	21

Target

$batch\_size = 3$

3	4	5	6
11	12	13	14
19	20	21	22

$seq\_len = 4$

Input	Target
2	3
2,3	4
2,3,4	5
2,3,4,5	6
10	11
10,11	12
10,11,12	13
10,11,12,13	14
18	19
18,19	20
18,19,20	21
18,19,20,21	22

How to choose batches?

- Larger batches
  - More stable gradients, smoother convergence
- Smaller batches
  - Noisier gradients, can help escape local minima
- Batch size
  - 32-128 for most language models

- Learning Rate Scaling:
  - Larger batches often require higher learning rates (why?)
  - Scale LR proportionally to batch size

How to choose sequence length?

- Longer sequences
  - provide more training data
  - Better long-range dependencies, more context
- Shorter sequences
  - Faster training, less memory
- Trade-off
  - Computational cost grows quadratically with sequence length in attention

Batch #0

$batch\_size = 3$

1	2	3	4
9	10	11	12
17	18	19	20

$seq\_len = 4$

$batch\_size = 3$

2	3	4	5
10	11	12	13
18	19	20	21

$seq\_len = 4$

Why Learning Rate Scheduler?

- Need to control the learning rate dynamically during training.
  - Improve convergence.
  - Adapt to different training phases (warmup, decay).

- Common Schedulers
  - Linear Decay
  - Cosine Decay

Warmup + Decay Strategy

- Warmup
  - Start with a small LR, gradually increase.
- Decay
  - Reduce LR after warmup to fine-tune learning.
- Why warmup + Decay?
  - Stabilizes training in early steps.
  - Prevents gradient explosion.
  - Helps large models converge smoothly.

Linear vs Cosine Decay

- Warmup
  - $lr(t) = lr_{base}\frac{t}{W}. \quad 0 \le t < W$
- Linear Decay
  - $lr(t) = \max\left(lr_{min}, lr_{base}\left(1 - \frac{t-W}{T-W}\right)\right) \quad W \le t \le T$
  - LR decreases linearly after warmup.
  - Simple and predictable.
- Cosine Decay
  - $lr(t) = \max\left(lr_{min}, lr_{base}\frac{1}{2}\left(1 + \cos\left(2\pi\, cycles\,\frac{t-W}{T-W}\right)\right)\right) \quad W \le t \le T$
  - LR follows a cosine curve.
  - Avoids local minima

•**total_steps**: Total training steps
•**warmup_ratio**: Fraction of steps for warmup.
•**base_lr**: Starting learning rate.
•**min_lr**: Floor value to prevent LR from going too low.

W = warmup_ratio × total_steps

Q3: Layer Norm

Ley $\mathbf{y}$ be the output of Post-LN and it can be written as

$$\mathbf{y} = \mathrm{LN}\big(\mathbf{x} + \mathrm{FFN}(\mathbf{x})\big).$$

Assume three input tokens with the corresponding embedding vectors $\mathbf{x}_i$, $i \in \{1, 2, 3\}$:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \qquad \mathbf{x}_2 = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 5 \\ 2 \\ 6 \end{bmatrix}$$

FFN is a two-layer MLP with ReLU:

$$\mathrm{FFN}(x) = W_2^T \,\mathrm{ReLU}(W_1^T x + b_1) + b_2,$$

with

$$W_1 = I_3, \quad b_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0 \end{bmatrix}, \quad W_2 = I_3, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

LayerNorm parameters (per feature):

$$\gamma = \begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix}, \qquad \beta = \begin{bmatrix} 0.5 \\ -0.5 \\ 1.0 \end{bmatrix}$$

Compute the output of Layer Normalization during training and inference.

Layer Normalization

Minibatch: $n = 5$ samples and $m = 4$ features

	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$		
$x_i^{(1)}$	2	80	400	0.5	μ_1	σ_1^2
$x_i^{(2)}$	4	90	300	0.7	μ_2	σ_2^2
$x_i^{(3)}$	6	70	500	0.4	μ_3	σ_3^2
$x_i^{(4)}$	8	85	600	0.6	μ_4	σ_4^2
$x_i^{(5)}$	10	95	200	0.8	μ_5	σ_5^2

$$\boldsymbol{x}_i = \begin{bmatrix} x_i^{(1)} \\ \vdots \\ x_i^{(n)} \end{bmatrix} \in \boldsymbol{R}^{n\times 1}$$

$$\overline{\boldsymbol{x}}_i \in \boldsymbol{R}^{n\times 1}$$

- For each sample with m features:
  - Calculate per-sample mean and variance:
    - $\mu_i = \frac{1}{m}\sum_{i=1}^{m} x_i^{(j)}, i = \{1,2 \ldots, n\}$
    - $\sigma_i^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_i^{(j)} - \mu_i\right)^2, i = \{1,2 \ldots, n\}$
  - Normalize the data
    - $\overline{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$, where ϵ is a small constant for numerical stability
  - Scale and shift with learnable parameters $\boldsymbol{\gamma} \in \boldsymbol{R}^{m\times 1}$ and $\boldsymbol{\beta} \in \boldsymbol{R}^{m\times 1}$
    - $\boldsymbol{y}_i = \boldsymbol{\gamma} \circ \overline{\boldsymbol{x}}_i + \boldsymbol{\beta}$
- Learnable parameters allows undoing of normalization if needed

Layer Normalization

Ley $\mathbf{y}$ be the output of Post-LN and it can be written as

$$\mathbf{y} = \mathrm{LN}\big(\mathbf{x} + \mathrm{FFN}(\mathbf{x})\big).$$

Assume three input tokens with the corresponding embedding vectors $\mathbf{x}_i$, $i \in \{1, 2, 3\}$:

$$\mathbf{x}_1 = \begin{bmatrix}1\\0\\2\end{bmatrix}, \qquad \mathbf{x}_2 = \begin{bmatrix}3\\1\\4\end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix}5\\2\\6\end{bmatrix}$$

FFN is a two-layer MLP with ReLU:

$$\mathrm{FFN}(x) = W_2^T\,\mathrm{ReLU}(W_1^T x + b_1) + b_2,$$

with

$$W_1 = I_3, \quad b_1 = \begin{bmatrix}0.5\\-0.5\\0\end{bmatrix}, \quad W_2 = I_3, \quad b_2 = \begin{bmatrix}0\\0\\0\end{bmatrix}.$$

LayerNorm parameters (per feature):

$$\gamma = \begin{bmatrix}2\\1\\0.5\end{bmatrix}, \qquad \beta = \begin{bmatrix}0.5\\-0.5\\1.0\end{bmatrix}$$

Compute the output of Layer Normalization during training and inference.

$$\boldsymbol{x}_1 + FFN(\boldsymbol{x}_1) = \begin{bmatrix}2.5\\0\\4\end{bmatrix}$$

$$\mu_1 = 2.166$$

$$\sigma_1 = 1.649$$

$$\frac{\boldsymbol{x}_1 - \mu_1}{\sigma_1} = \begin{bmatrix}0.202\\-1.31\\1.11\end{bmatrix}$$

$$LN\big(\boldsymbol{x}_i + FFN(\boldsymbol{x}_i)\big) = \boldsymbol{\gamma} \circ \left(\frac{\boldsymbol{x}_i - \mu_i}{\sigma_i}\right) + \boldsymbol{\beta}$$

$$LN\big(\boldsymbol{x}_1 + FFN(\boldsymbol{x}_1)\big) = \begin{bmatrix}2\\1\\0.5\end{bmatrix} \circ \left(\frac{\boldsymbol{x}_1 - \mu_1}{\sigma_1}\right) + \begin{bmatrix}0.5\\-0.5\\1.0\end{bmatrix} = \begin{bmatrix}0.904\\-1.813\\1.555\end{bmatrix}$$

Layer Normalization

Ley $\mathbf{y}$ be the output of Post-LN and it can be written as

$$\mathbf{y} = \mathrm{LN}\big(\mathbf{x} + \mathrm{FFN}(\mathbf{x})\big).$$

Assume three input tokens with the corresponding embedding vectors $\mathbf{x}_i$, $i \in \{1, 2, 3\}$:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \qquad \mathbf{x}_2 = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 5 \\ 2 \\ 6 \end{bmatrix}$$

FFN is a two-layer MLP with ReLU:

$$\mathrm{FFN}(x) = W_2^T \, \mathrm{ReLU}(W_1^T x + b_1) + b_2,$$

with

$$W_1 = I_3, \quad b_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0 \end{bmatrix}, \quad W_2 = I_3, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

LayerNorm parameters (per feature):

$$\gamma = \begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix}, \qquad \beta = \begin{bmatrix} 0.5 \\ -0.5 \\ 1.0 \end{bmatrix}$$

Compute the output of Layer Normalization during training and inference.

$$\boldsymbol{x}_2 + FFN(\boldsymbol{x}_2) = \begin{bmatrix} 6.5 \\ 1.5 \\ 8 \end{bmatrix}$$

$$\mu_2 = 5.33$$

$$\sigma_2 = 2.77$$

$$\frac{\boldsymbol{x}_2 - \mu_2}{\sigma_2} = \begin{bmatrix} 0.4198 \\ -1.379 \\ 0.959 \end{bmatrix}$$

$$LN\big(\boldsymbol{x}_i + FFN(\boldsymbol{x}_i)\big) = \boldsymbol{\gamma} \circ \left(\frac{\boldsymbol{x}_i - \mu_i}{\sigma_i}\right) + \boldsymbol{\beta}$$

$$LN\big(\boldsymbol{x}_2 + FFN(\boldsymbol{x}_2)\big) = \begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix} \circ \left(\frac{\boldsymbol{x}_2 - \mu_2}{\sigma_2}\right) + \begin{bmatrix} 0.5 \\ -0.5 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1.339 \\ -1.879 \\ 1.479 \end{bmatrix}$$

Layer Normalization

Ley $\mathbf{y}$ be the output of Post-LN and it can be written as

$$\mathbf{y} = \mathrm{LN}\big(\mathbf{x} + \mathrm{FFN}(\mathbf{x})\big).$$

Assume three input tokens with the corresponding embedding vectors $\mathbf{x}_i$, $i \in \{1, 2, 3\}$:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \qquad \mathbf{x}_2 = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 5 \\ 2 \\ 6 \end{bmatrix}$$

FFN is a two-layer MLP with ReLU:

$$\mathrm{FFN}(x) = W_2^T \, \mathrm{ReLU}(W_1^T x + b_1) + b_2,$$

with

$$W_1 = I_3, \quad b_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0 \end{bmatrix}, \quad W_2 = I_3, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

LayerNorm parameters (per feature):

$$\gamma = \begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix}, \qquad \beta = \begin{bmatrix} 0.5 \\ -0.5 \\ 1.0 \end{bmatrix}$$

Compute the output of Layer Normalization during training and inference.

$$\boldsymbol{x}_3 + FFN(\boldsymbol{x}_3) = \begin{bmatrix} 10.5 \\ 3.5 \\ 12 \end{bmatrix}$$

$$\mu_3 = 8.66$$

$$\sigma_3 = 3.70$$

$$\frac{\boldsymbol{x}_3 - \mu_3}{\sigma_3} = \begin{bmatrix} 0.4949 \\ -1.394 \\ 0.899 \end{bmatrix}$$

$$LN\big(\boldsymbol{x}_i + FFN(\boldsymbol{x}_i)\big) = \boldsymbol{\gamma} \circ \left(\frac{\boldsymbol{x}_i - \mu_i}{\sigma_i}\right) + \boldsymbol{\beta}$$

$$LN\big(\boldsymbol{x}_3 + FFN(\boldsymbol{x}_3)\big) = \begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix} \circ \left(\frac{\boldsymbol{x}_3 - \mu_3}{\sigma_2}\right) + \begin{bmatrix} 0.5 \\ -0.5 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1.489 \\ -1.894 \\ 1.449 \end{bmatrix}$$

Why Pre-LN preferred over Post-LN?

On Layer Normalization in the Transformer Architecture

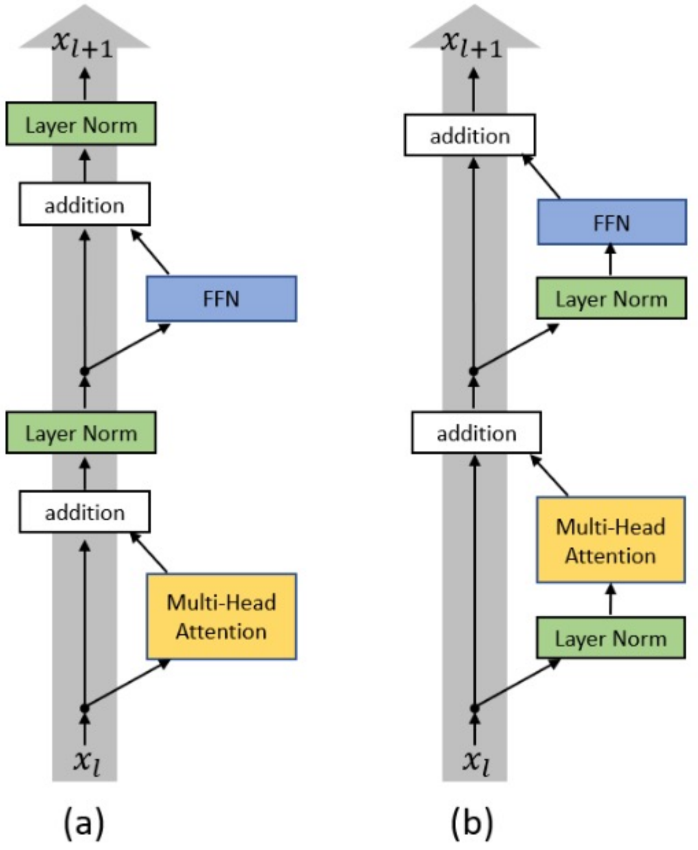
Ruibin Xiong[†*1 2] Yunchang Yang[*3] Di He[4 5] Kai Zheng[4] Shuxin Zheng[5] Chen Xing[6] Huishuai Zhang[5]
Yanyan Lan[1 2] Liwei Wang[4 3] Tie-Yan Liu[5]

https://arxiv.org/pdf/2002.04745

Post-LN

```python
class TransformerBlock(nn.Module):
    def __init__(self, d_model, num_heads, d_ff, dropout=0.1):
        super().__init__()
        self.attention = MultiHeadAttention(d_model, num_heads)
        self.feed_forward = PositionwiseFeedForward(d_model, d_ff)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, mask=None):
        attn_out = self.attention(x, x, x, mask)
        x = self.norm1(x + self.dropout(attn_out))
        ff_out = self.feed_forward(x)
        x = self.norm2(x + self.dropout(ff_out))
        return x
```



(a) Post-LN Transformer layer; (b) Pre-LN Transformer

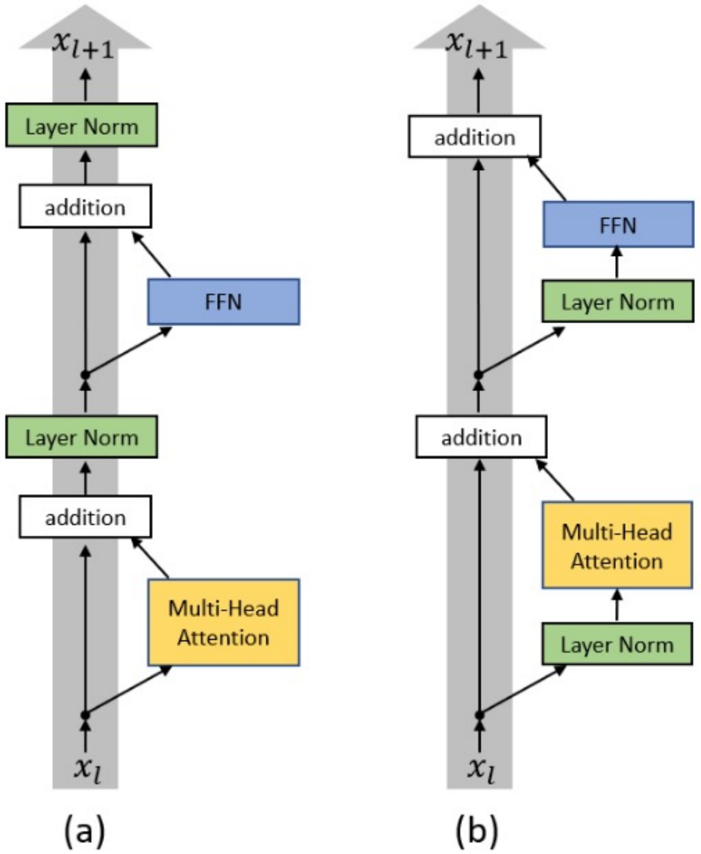
Pre-LN

```python
class TransformerBlock(nn.Module):
    def __init__(self, d_model, num_heads, d_ff, dropout=0.1):
        super().__init__()
        self.attention = MultiHeadAttention(d_model, num_heads)
        self.feed_forward = PositionwiseFeedForward(d_model, d_ff)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, mask=None):
        # Pre-LN attention: normalize input, apply attention, then residual add
        x_norm = self.norm1(x)
        attn_out = self.attention(x_norm, x_norm, x_norm, mask)
        x = x + self.dropout(attn_out)

        # Pre-LN FFN: normalize current x, apply FFN, then residual add
        x_norm = self.norm2(x)
        ff_out = self.feed_forward(x_norm)
        x = x + self.dropout(ff_out)

        return x
```

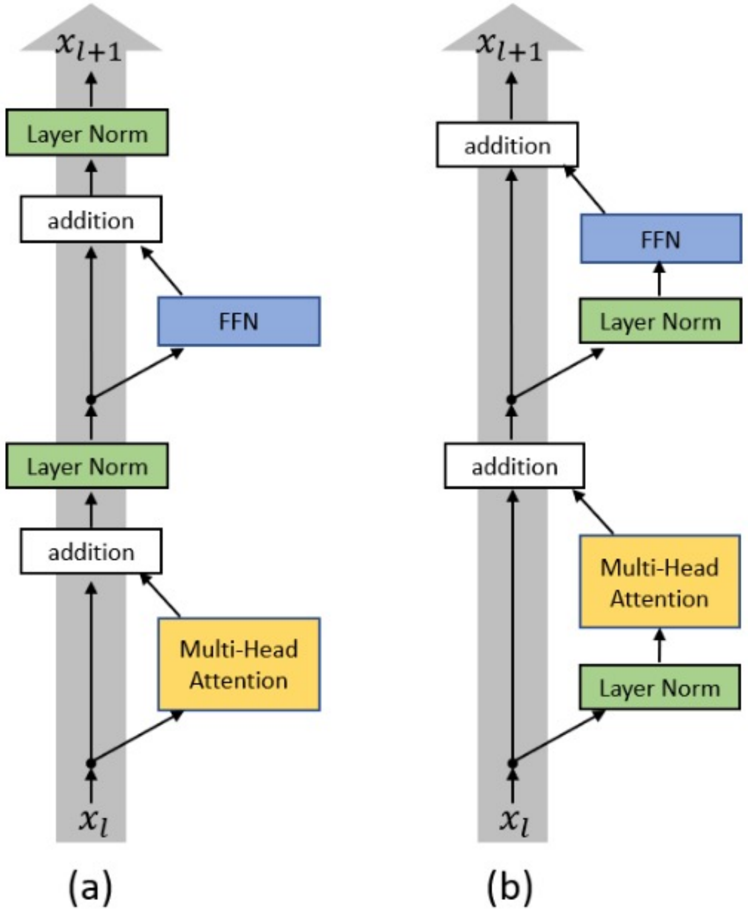


(a) Post-LN Transformer layer; (b) Pre-LN Transformer

Why Pre-LN preferred over Post-LN?

- Post-LN block
  - Attention sublayer:
    - $y = \mathrm{LN}\ (x + \mathrm{Attn}(x))$
  - FFN sublayer:
    - $y = \mathrm{LN}\ (x + \mathrm{FFN}(x))$
- Pre-LN block
  - Attention sublayer:
    - $y = x +\ (\mathrm{Attn}(\mathrm{LN}\ (x))$
  - FFN sublayer:
    - $y = x +\ (\mathrm{Attn}(\mathrm{LN}\ (x))$

Gradient Highway



Gradient Highway

Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

Why Pre-LN preferred over Post-LN?

$$\frac{\partial y}{\partial x} = \frac{\partial LN}{\partial x}\left(I + \frac{\partial Att}{\partial x}\right)$$

$$\frac{\partial y}{\partial x} = \frac{\partial LN}{\partial x}\left(I + \frac{\partial FFN}{\partial x}\right)$$

Gradients via residual paths are multiplied by
gradients of LN block

It can shrink or distort the gradients

$$\frac{\partial y}{\partial x} = I + \frac{\partial Attn}{\partial LN}\frac{\partial LN}{\partial x}$$

$$\frac{\partial y}{\partial x} = I + \frac{\partial FFN}{\partial LN}\frac{\partial LN}{\partial x}$$

Clean ‘identity’ path for gradient or gradient highway via residual paths

Converges robustly and allows higher learning rates

Gradient Highway

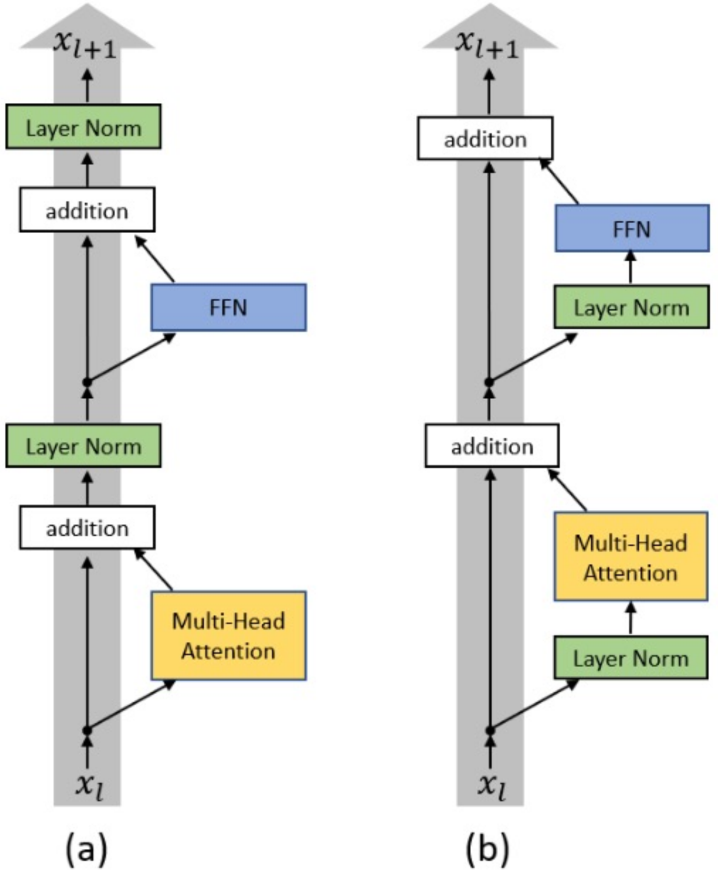


Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

Pre-LN

- Learning Rate scheduler
    - Can use larger learning rate compared to post-LN
    - No warmup needed
        - 'warmup_ratio': 0.0,

How to improve performace?

- Large d_model. (start with small 128 and increase it to 256)
- Increase sequence length (start with 100 and increase it to 256)
  - Larger context
- Different activation (start with ReLU and change it to GELU)
- Pre-LN
- Lower drop out
- Stop the training once the validation loss plateaus
- If validation loss increases after above changes, revert the change…it’s a sign of overfitting